FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Computer-Vision-based surveillance of Intelligent Transportation Systems

João Neto

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Rosaldo Rossetti, PhD

Second Supervisor: Diogo Santos, MsC

June 27, 2017

Computer-Vision-based surveillance of Intelligent Transportation Systems

João Neto

Mestrado Integrado em Engenharia Informática e Computação

June 27, 2017

Abstract

Traffic management is one of the most important and crucial tasks in modern cities, due to the increasing number of vehicles in the under dimensioned road networks. To aid in this task multiple solutions exist that simulate the networks behaviour taking for input data from traditional counting methods such as magnetic induction loops. These devices present flaws and are expensive to implement. With the decreasing cost of video cameras and the wide-spread use of computer vision, their use became a viable alternative to analyse traffic.

In this project we tackle the problem of automatic traffic surveillance using video cameras, developing a video analytics processor that will perform detection and classification tasks over gathered data. This processor was integrated in a solution under development that treats videos from multiple sources and provides computer vision as an on-line service.

A novel approach to the vehicle classification process is presented, based on the use of a fuzzy set. To illustrate the proposed approach, the detection and classification implemented were tested with different cameras in different scenarios, showing promising results.

Acknowledgements

Now that this period of my life is upon its dawn, it is time to thank everyone who accompanied me through this journey which has seen me grow both as an individual and as a professional. Without your support this adventure would not have been the same.

To the $\rho 7$ Boyz, João Pereira, José Pinto, Henrique Ferrolho, Luís Reis and João Dias, as well as Sara Paiva and Beatriz Baldaia, the residents of Lab I120, thank you for all the good moments.

To my parents and my brother for always being there for me, offering sage advice and true friendship. Thank you to my girlfriend Cátia Matos, who on a daily basis helped me face the toughest problems and overcome the greatest challenges.

To all my friends who showed a great degree of interest with this project, always ready to help, João Paiva, Jorge Wolfs, Sara Serra, Margarida Borges and Tomás Matos, thank you all.

I would also like thank my supervisor Prof. Dr. Rosaldo Rossetti for his guidance throughout the project as well as shedding some light into my professional life, and my second supervisor Diogo Santos for helping me take my first steps in the world of computer vision.

João Neto

"The field is lost
Everything is lost
The black one has fallen from the sky and the towers in ruins lie
The enemy is within, everywhere
And with him the light, soon they will be here
Go now, my lord, while there is time
There are places below"

Hans Jürgen Kürsch

Contents

List of Figures

LIST OF FIGURES

List of Tables

LIST OF TABLES

Abbreviations

2D	Two Dimensions
3D	Three Dimensions
ACM	Association for Computing Machinery
ADT	Abstract Data Type
ANDF	Architecture-Neutral Distribution Format
API	Application Programming Interface
ATON	Autonomous Transportation Agents for On-scene Networked incident manager
BGSub	Background Subtracter
CUDA	Compute Unified Device Architecture
FFmpeg	Fast Forward Moving Pictures Expert Group
HSV	Hue, Saturation and Value
IEEE	Institute of Electrical and Electronics Engineers
ITS	Intelligent Transport Systems
LIACC	Laboratório de Inteligência Artificial e Ciência de Computadores
MIT	Massachusetts Institute of Technology
MRI	Magnetic resonance imaging
OpenCV	Open Source Computer Vision
RGB	Red Green Blue
SAKBOT	Statistical and Knowledge-BasedObject Tracker
YACCLAB	Yet Another Connected Components Labelling Benchmark
CAGR	Compound Annual Growth Rate
OpenCV	Open Source Computer Vision Library

Chapter 1

Introduction

1.1 Context

Traffic management in cities is a vast area as it studies the planning and control of the road network, with all the tasks associated with them. As the cities tend to evolve following the concept of *smart cities*, the management of how its people move is a prime area where the information technologies are being applied. With the increase in the number of vehicles using the roads [Nav00], this need to improve the methods of traffic management rise even more, and the development of multiple projects around the world around this issue comes as a confirmation of both it's importance and the pertinence of the work being developed at LIACC in regards to this topic, the development of MasterLab, a dashboard for the management of traffic and transportation [ZRC14] [PRO10].

With the decreasing costs of cameras for video surveillance, the number of units installed around the world is rising, passing the 245 million mark in 2014 [Jen15], over 65% of that number being from Asia. With this number of cameras placed globally the amount of data being collected every day is too large to be humanly processed, and thus the need to create autonomous processors arises. The market for automatic analysis of video is expected to be worth 11.17 Billion USD by 2022 [Rep17], with the facial recognition area expected to have the highest CAGR (Compound Annual Growth Rate) due to the potential related to the security applications, even going as far as replacing airport checks in Australia by 2020 [Koz17].

The usage of computer vision to analyse traffic conditions has been around for some time now, with earlier works dating to 1990, such as the work by Rourke et. al [RBH90] where the advantages of using video analysis in this area are listed, as well as some issues that the community is still facing. Since then the field has been explored by many authors, applied to applications ranging from vehicle counting [CBMM98] [BMCM97] to incidents detection. Some work has been already developed at LIACC regarding this area of application of computer vision, namely an approach to use uncontrolled video streams found on web to perform traffic control in [LRB09].

Armis-Group is a company based in Porto that builds software in the areas of Information Management, Digital Sports and Intelligent Transport Systems. They are now developing two projects in the area of traffic analysis, Drive and Next [Adm17]. Drive is an innovative Traffic

Management and Control solution, flexible and with an easy and fast set-up. Next is a complement to Drive that analyses, simulates and predicts traffic and events data. Due to the data simulation it is a valuable aid to the city operators decision making.

1.2 Project

The project for this dissertation was part of a collaboration between LIACC and Armis-Group. This collaboration aims to develop solutions in the field of Traffic Management, taking advantage of the laboratory's resources and the company's expertise and experience in the area. The objective of the project was to design and develop a software module capable of analysing video and extracting relevant events on the roads as well as performing counting and basic classification of vehicles.

In the scope of the mentioned collaboration, LIACC developed a set of applications that form the "Video Server", a software package that aims to provide Computer Vision as a service, treating input streams as per user request. The analytics module developed during this dissertation work is part of its architecture.

The resulting module of the project can be used to analyse the traffic flow and to feed that information to simulation systems such as described in [PRK11] [ARB15] [TARO10] [FRBR09]. Besides this, the project can be used as a complement to other counting mechanisms such as the ones presented in [SRM+16] and [BAR15].

1.3 Aim and Goals

The main goal of the dissertation is then to implement a solution that fits the needs of the project as stated above, detecting events in a short window of time to allow for fast response from the authorities. This can be further broken down into more specific tasks to allow for a better understanding of the work ahead. The specific goals are as follows.

- To contribute for the effective integration of the implemented solution of this dissertation into their Armis Group commercial product;
- To implement solutions so as to perform the following tasks:

  Clear the image from noise and other unwanted features;

  Find objects of interest;

  Classify detected objects;

  Detect relevant events.
- To test the implementation in a real scenarios.

1.4 Dissertation Structure

This document is divided into four chapters. Chapter 2 reviews the beginnings of computer vision, how it evolved and it's applications, followed by an overview of basic image treatment techniques into more advanced ones, as well as presenting the more relevant work in the area of computer vision applied to traffic analysis. Chapter 3 details the techniques applied during the development in a more practical approach. In Chapter 4 the architecture of the implemented solution as well as the results of the work are presented and discussed. Finally, in Chapter 5 conclusions of the work and the contributions of the dissertation are described, as well as where to take the project next.

Chapter 2

Computer Vision and Traffic Analysis - A literature review

2.1 History of Computer Vision

Computer vision appeared as an area of investigation around the mid 60's at the MIT by Professor Larry Roberts, whose doctorate thesis focused on methods to extract 3D information from a 2D image [Hua96] in order to reconstruct entire scenes from the geometry gathered. This area is now considered by the ACM as a branch of artificial intelligence according to the 2012 ACM Computing Classification System [ACM12]. From this point onward scientists began applying the techniques developed in multiple fields.

The application of computer vision in the manufacturing industry was amongst the first ones to be explored. Due to the sheer number of robots working in the assembly lines of the factories that needed to be improved, as noted by Stout in 1980 [Sto80]. These robots were becoming outdated due to the lack of interaction with the environment. This meant that for a robot to work with a part in an assembly line it needed to be placed in a pre-determined position, and any deviation could mean that the line needed to be stopped. Michael Baird relates in [LB78] a computer vision system developed to inspect circuit chips rotation deviation in a welding base, indicating to the welder that the chip needed to be adjusted. Computer vision was also used as a tool to automatically analyse weather satellite images [BT73] and multidimensional medical images [Aya98], with applications being developed in these areas being now common in our daily life.

2.1.1 Computer Vision in Intelligent Transport Systems

Regarding intelligent transport systems (ITSs), the use of computer vision to aid in the analysis of traffic has been increasing in the last years due to the decreasing costs of hardware, both cameras, storage and processing power, as well as the growing knowledge to extract useful data from the video gathered. In contrast to the high installation and upkeep costs of other traffic control tools such as Inductive Loop Detectors and Microwave Vehicle Detectors, applying computer vision to handle these tasks is a profitable option for entities in charge of the analysis of this data.

One of the first works applying computer vision to analyse traffic, which was published in 1984 [DW84] and detected and measured movement in a sequence of frames. Since then, multiple fields of study have been created, not only for the measuring of traffic, as explored in [LKR+16], [LRB09], [BVO11] and [HK12] where the authors evaluate methods to analyse traffic in urban environments, but also for the analysis of the environment around a self-driven vehicle, reading traffic signs using multiple approaches using convolutional neural networks [SS11] or using bag-of-visual-words [SLZ16], or to automatize the parking process as discussed in [HBJ+16]. Recently some studies have surfaced where the authors discuss the analysis of passenger numbers and behaviour inside vehicles, in order to enforce traffic laws [LBT17].

However our work upon this topic is focused on the aspect of traffic analysis. In order to understand what we need to accomplish, it is convenient to study what composes a typical traffic scene. Usually the scenes are viewed from a top-down perspective that places the cars against the road as background, as seen in Figure 2.1. The vehicles have a roughly regular rectangular shape when viewed from the top, with little variation when the camera is rotated, however their textures vary heavily, making it difficult for a detector to work based on the vehicles image representation [BP98]. However, depending on the camera position there can be object occlusion by other objects or scene components, which makes it difficult to detect and track vehicles relying solely on subtraction based segmentation techniques.

Figure 2.1: Typical High-Way Traffic Scene

The scenes also have varying light according to the time of the day, and proposed solutions need to adapt to these changes as fast as possible, otherwise data might be lost. Other weather conditions can also interfere with the analysis, such as fog and rain, and need to be addressed when designing solutions.

2.1.2 Our Challenge

The "Analytics Server" module of the "Video Server" project had a list of requirements that needed to be addressed:

- Detect relevant events:

Wrong way driving;

Fallen objects on the road;

Prohibited zone entering;

Suspicious camera approximation

- Count and classify vehicles into light and heavy categories

The following sections of this document will explore the theoretical basis on which our solution was built, aimed at tackling these specific points.

2.2 Image Treatment

This section will review some of the proposed methods to improve image quality before or during processing by removing noise, extracting unwanted features or enhancing relevant features for the processes involved.

2.2.1 Blur

In order to blur an image one needs to convolve it using a special kind of filter. The process of convolution in image processing consists in the application of a filter to every pixel of an image, returning a new image where each pixel is the result of the function to the pixel in the same position in the previous image. As we can see in Figure 2.2 the application of a 3*3 mask, where all 9 values have the value 1/9, results in a image where each pixel value equals the mean of itself and surrounding neighbours in the original image.

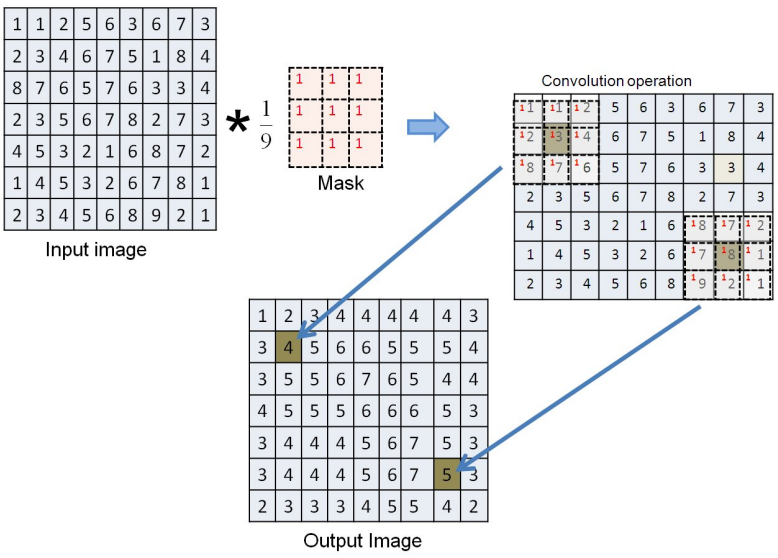


Figure 2.2: Convolution Example [Lab17]

While this is the simplest blur processing we can apply to an image, there are more complex ones that yield better results. The one most often used in image processing is the Gaussian Blur, which uses a square filter with values calculated with the equation in 2.1 from [SS01].

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.1}$$

This equation returns a filter consisting of concentric circles that results in a smooth blurring of the images due to the application of the same power of blurring from equidistant pixels. The function can be tuned to the needs of the process through the σ value, returning a sharper or softer blur filter as seen in Figure 2.3.

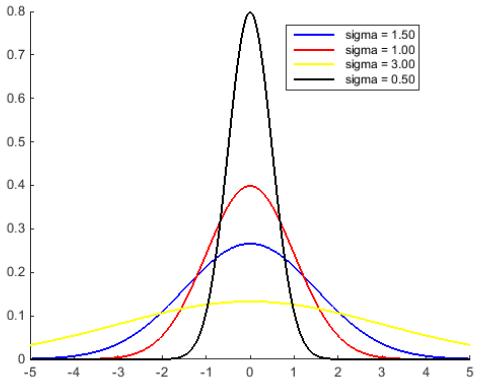


Figure 2.3: Gaussian Function Variation with Sigma Value

This kind of filters have multiple uses in computer vision. Blurring an image removes the grain noise it contains, creating more even surfaces that ease the process of image processing, as well as making edges easier to detect with edge detection algorithms.

2.2.2 Mathematical Morphology

Mathematical Morphology is a set theory, offering solutions to multiple image processing problems, where the sets are composed of tuples corresponding to the 2D coordinates of each pixel in the image. These sets identify the white or the black pixels in a binary image, depending on the convention adopted, or the intensity of the pixel in a grey-scale image [Dou92]. During this chapter we will consider that all images are binary and that the pixels to process are black in a white background. This theory provides a set of operators that can be used to detect convexity and connectivity of shapes in an image, enhancing features, detecting edges and removing noise.

The operators are defined by a structuring element, also known as kernel, a set of coordinates represented in a binary image, usually much smaller than the image to process. The centre of the kernel is not in the top left corner as in most images but rather in the geometrical centre, meaning that there some of the pixels have negative coordinates. The kernel shapes in Figure 2.4 are the

ones typically used for most applications, but the process allows the user to design their own if needed.

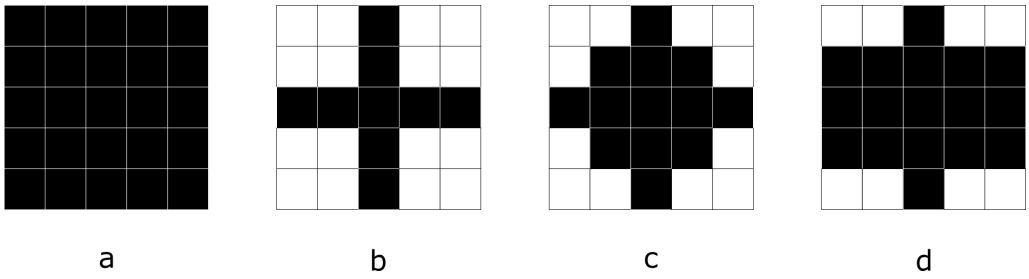


Figure 2.4: Morphology Operators typical structuring elements (5x5)
a) Rectangular b) Cross c) Diamond d) Elliptical

According to Gonzalez and Woods [GW92] an operator is defined by an image and a structuring element, and to run it the origin point of the element needs to be placed at every pixel of the image. At each of these locations a verification is done depending on the operation being performed and an output image is created using the independent output of these operations.

The two most basic operations are erosion and dilation, essential to morphological processing. The erosion operator translates the structuring element to every black pixel of the image and in each position checks if all the active pixels of the kernel match with black pixels in the image. If it is true then the returned pixel is black, otherwise it is white. An example of the usage of such an operator can be seen in Figure 2.5. These operators are mainly used to remove noise from binary images or to separate objects to enable individual labelling and counting.

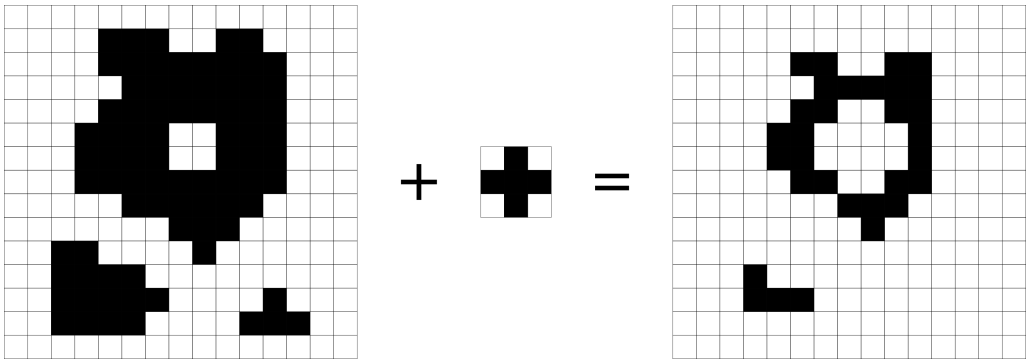


Figure 2.5: Result of the Erosion operator using a cross 3x3 kernel

The dilation operator works by setting to black all the pixels that match the position of the translated kernel's pixels, resulting in the closing of holes in the image and the enhancing of small features. Results can be seen in Figure 2.6. Using a non-symmetrical kernel the dilation will be directional, enabling the user to fine tune the process to their needs.

The dilation and erosion operators can be combined to form the opening and closing operators. An opening operator is formed by an erosion followed by a dilations using the same structuring

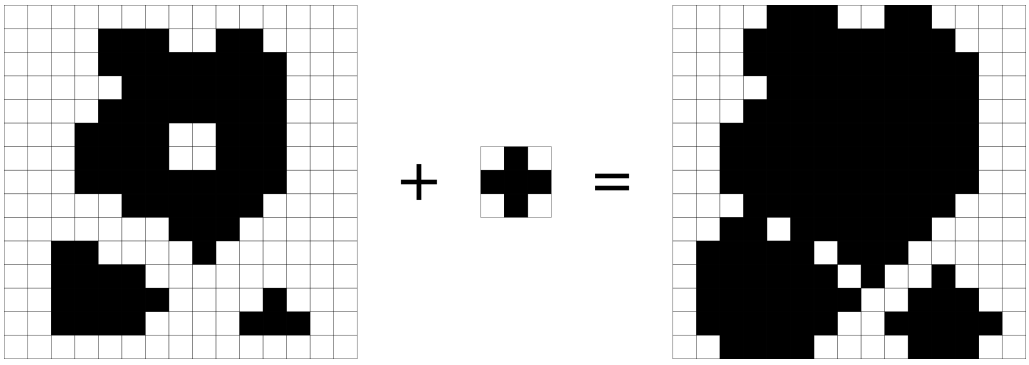


Figure 2.6: Result of the Dilation operator using a cross 3x3 kernel

element. This results in a less destructive erosion, preserving some of the image details as seen in Figure 2.7. The closing operator is formed by a dilation followed by an erosion and the results can be seen in Figure 2.8. It is commonly used to remove salt noise and to extract objects of a particular shape as long as they all share the same orientation [FPWW03].

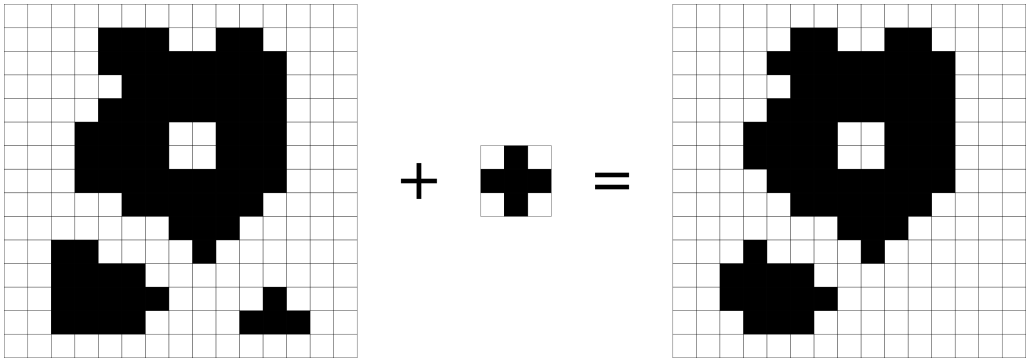


Figure 2.7: Result of the Opening operator using a cross 3x3 kernel

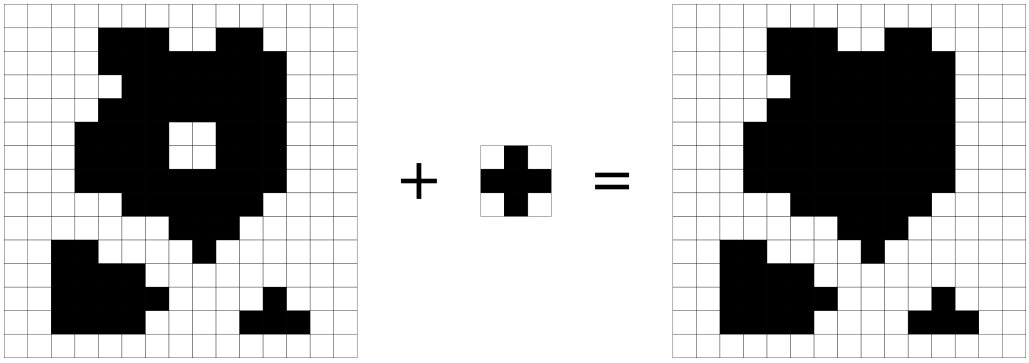


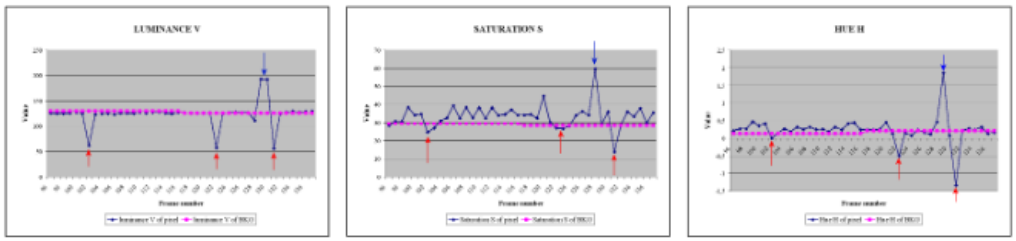
Figure 2.8: Result of the Closing operator using a cross 3x3 kernel

2.2.3 Shadow Removal

Since shadows can be detected as foreground during the segmentation process, a process to remove them is important in certain scenarios. We will now describe some of the more interesting proposed methods to perform this task in this section.

In [PMGT01] Prati et al. discussed how critical the shadow removal process is to traffic analysis, and they proceed to compare two ITS management systems, SAKBOT (Statistical and Knowledge-BasedObject Tracker) and ATON(Autonomous Transportation Agents for On-scene Networked incident manager), developed in Italy and the United States of America respectively.

SAKBOT adopts a method presented in [CGPP00], which uses the chrominance information, converting pixel colours from RGB to HSV space, as it most closely relates to the human perception of colour. The process then evaluates how the scene colours components change due to the passing of vehicles and shadows, as seen in Figure 2.9. From this we can extract that a shadow darkens a pixel and saturates its colour, and is the difference between an object point and a shadow point. The model then concludes that a point is considered as shadow if:



(a) Luminance (b) Saturation (c) Hue

Figure 2.9: HSV color space components change due to shadows. Red arrows indicate a shadow passing on the point. Blue arrow indicates a vehicle passing [PMGT01] ©[2001] IEEE

- The ratio between the luminance of the image and the background is between two thresholds (α and β)
- The difference between the saturation of the image and the background is below a threshold (τ_S)
- The absolute difference between the hue of the image and the background is below a threshold (τ_H)

The β value adjusts the tolerance to noise in the image to be considered as shadow and α takes into account the "power" of the shadow, how strong the light source is. τ_S and τ_H are considered by the authors to be harder to assign, and thus assigned empirically. This method main advantage is it's capability to detect moving shadows.

The method used by ATON is described in [MCKT00] and begins by identifying three separate sources of information to detect shadows and objects:

- Local information, the appearance of an individual pixel
- Spatial information, objects and shadows are compact regions
- Temporal information, the location of shadows and objects can be predicted from previous frames

The local information can be used to create a background model, both the mean and variance of the colour for each pixel when shadowed and not shadowed. In the segmentation process the values in the image are compared to the mean value of the background and if significantly different it is assigned a probability to belong to the background, foreground or shadow. The probability values are 0.3, 0.4 and 0.4 respectively. In the next iterations the neighbour pixels probabilities are considered in order to take advantage of the spatial transformation. The iterative process stops when there are no relevant changes, usually after the third iteration. Morphological operators are then computed over the output of the process to close unwanted openings in the regions. ATON does not use temporal information as of the time of the publication of the work. This process can identify with near 90% accuracy shadows and object accuracy.

2.3 Foreground Segmentation

Image segmentation is the process through which an image is separated into its different regions according to the desired output, usually objects of interest. These regions are composed by pixels that have a common characteristic [SS01] dependant on the desired result. There are several ways one can accomplish an adequate segmentation of an image, and the most relevant ones to our purpose are described below.

2.3.1 Background Subtraction

Background Subtraction is a segmentation technique based on the analysis of the difference of consecutive frames and use that information to create a background model, a representation of what the image looks like without any moving objects present. Given the nature of the process, cameras must be static and although some research has been made to overcome this issue [LCC12] [SJK09] [ZY14], this is beyond the scope of the project, due to the requirements given.

There are however a number of different ways to implement the background subtraction developed across the years with increasing segmentation accuracy and computational performance. Piccardi presented an overview of the most relevant ones in [Pic04], aiming to present each method's strengths and weaknesses. From this work we can present a list of the algorithms considered for implementation in the project.

2.3.1.1 Frame Differencing

This is the simplest approach to the problem as it only considers two frames at a time, making it only work in certain scenarios where the speed of the objects and frame rate of the camera allow

it. In this model a pixel is considered as foreground if it satisfies the equation presented in 2.2, and since the only parametrizable value is the Threshold, the whole segmentation process is very sensitive to any changes in this value.

$$|frame_i - frame_{i-1}| >= Threshold \tag{2.2}$$

2.3.1.2 History Based Background Model

In order to make this segmentation method more robust, in the early 2000's Velastin [LV01] and Cucchiara [CGPP03] proposed a mathematical model to take into consideration past frames when calculating the Background Model of the scene. The most simple version of the improved algorithm used a running average of the past frames as seen in 2.3. This eliminates the need to store the frames in memory as the new average can be calculated using the previous results.

$$BackgroundModel_{i+1} = \alpha * Frame_i + (1 - \alpha) * BackgroundModel_i \tag{2.3}$$

Instead of the running average one can use the median value of the last n frames, improving the method reaction to outlying values. In both however, the history can be just the n frames or a weighted average where recent frames have more weight. Both these methods cannot cope with intermittent changes on the background, such as moving leaves against a building facade, and to solve this problem a new approach was proposed, the Mixture of Gaussians which models each pixel according to a mixture of Gaussian distributions, usually 3 to 5, but in later research [Ziv04] a method was developed to calculate the number of distributions needed on a per pixel basis.

2.3.2 Feature Tracking

When the literature mentions features it is referring to interesting parts of the image that can be detected and matched in multiple different images of the same scene. The process of detecting these features is a basic step in many computer vision applications, as they allow for a broad comprehension of the scene being observed. These features can take the shape of:

- Edges - A boundary in an image (Further explored in the next section)
- Corners or Interest Points - Points in the image that have distinctive neighbourhood, making them good candidates for tracking
- Blobs or Regions of Interest - Connected areas of the image that share a property like colour or intensity

The OpenCV implementation of *goodFeaturesToTrack* is based on the work of Jianbo Shi et al. [ST94] that tackles the problem of selecting features that can be tracked consistently across

the scene, usually feature points that correspond to physical points in the scene. It does so by calculating a value they named *dissimilarity* that quantifies how much the feature appearance has changed between the first and the current frame. When this value grows past a certain threshold, the point is discarded and no longer tracked.

The two main improvements presented in this paper regarding this problem are the fact that the authors allow for both linear warping and translation regarding the image motion model, and the second being a numerically superior method for calculating the *dissimilarity* value when compared to existing work. The other main contribution is a new value to measure how good a feature is, beside the already common *interest* and *cornerness*. This value is the *texturedness* and is a result of the authors findings that features with good texture are more easily tracked.

2.3.3 Edge Detection

Detecting the edges in an image is an important step into the understanding of its contents, allowing segmentation based on surfaces and a spatial perception of the scene. The work presented by Marr and Hildreth [MH80] defines edges as part visual and part physical and proceeds to show that the two are interconnected. The first step of the proposed process is to calculate the zero-crossings of equation 2.4, the Laplacian of a two dimensional Gaussian applied to different scales of the image, which returns both the points of intensity change and the slopes of the function at that point.

$$\nabla^2 G(x,y) * I(x,y) \tag{2.4}$$

What the authors noted was that if an edge was detected at a certain scale it appears also at adjacent scales, which they called "spatial coincidence assumption". This theory can be reversed, if there is a convergence of edges on multiple scales there is a real physical edge, and based on this, if we find zero-crossings on neighbour scales, we can assume there is a real edge in that region. From here multiple implementations of the algorithm appeared, the most used one being presented below, the Canny Edge Detector using a Sobel filter. Although this was among the first implementations of an edge detector it is the most rigorously defined one and is still considered to be state of the art.

The Sobel filter is used to create an image that accentuates the edges of an original image. This process works by convolving two kernels with the image, one for the horizontal and the other for the vertical direction, that return the intesity change in that pixel for each one. The filters can be seen in 2.5 as they were presented by Irwin Sobel [Sob89].

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}, G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2.5}$$

Using G_x and G_y it is possible to retrieve the regions of the image that contain edges by calculating the magnitude of the gradient vector as the square root of the sum of the squares of both components and thresholding these values. One of the features of this process is the ability to retrieve the direction of the edge from these values, using the inverse tangent operation.

The Canny Edge Detector uses the result of the Sobel filter as input and filters the most relevant edges as well as thinning them to 1-pixel wide lines. To accomplish this John Canny [Can86] used an optimization process comparing the value of the gradient magnitude to the one of the neighbour pixels along the direction returned by the Sobel filter, and keeping only the one with the largest value along the edge.

2.3.4 Object Based

Object Based segmentation relies on the a priori knowledge of the geometry of the objects we want to extract from the image. It is mainly used in medical applications such as in [SMG+93] where it is used to extract a model of the brain surface from MRI scans. To be able to use this approach to solve our issue of vehicle detection we would need detailed models of all the vehicles that can possibly pass through the scene we are analysing, which is not feasible, and thus this method was not further explored.

2.4 Object Detection

In this section we will analyse processes to detect objects of interest in images, a necessary step of any traffic surveillance system.

2.4.1 Connected Component Labelling

Connected component labelling is a process by which the subsets of connected components are labelled according to the user needs. We will explore some of the most common implementations and how they evolved over time.

The one-pass solution presented by Abubaker [AQIS07] is a fast and simple method to implement. It begins by labelling the first pixel of the image to 1, if it is a background pixel, skip to the next one, otherwise add it to a queue. While there are elements in the queue remove the one at the top and analyse its neighbours, if they are in the foreground add them to the queue with the current label. When there are no more elements in the queue increase the label number by one and proceed to the next unexplored pixel.

The two-pass solution presented by Shapiro et al. [SS01] uses the bi-dimensionality of the image data to transverse the pixels. The process, as the name implies, iterates over the image two times, the first time to assign temporary labels and the second to reassign these labels to the smallest ones available. It uses a neighbouring table to know what labels are adjacent to each other.

- First pass

Perform a Raster Scan to analyse all the pixels

For each pixel, if it belongs to the foreground

Get all its neighbours (four or eight depending on the connectivity assumed)

Assign the smallest label from the neighbours

Add the the previous and the new label to the neighbouring table

If there are no neighbour pixels add a new label

- Second pass

Perform a Raster Scan to analyse all the pixels

For each pixel assign the lowest label from the neighbouring table

The process is illustrated in Figure 2.10.

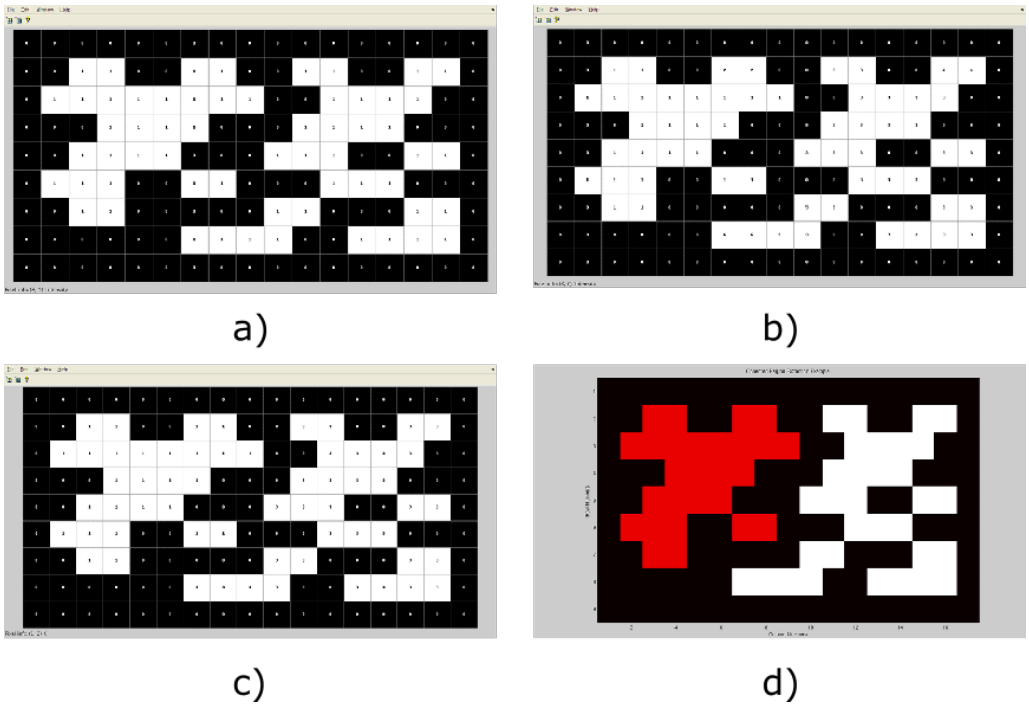


Figure 2.10: Graphical example of the two-pass algorithm [Wik17]
a) Original Image b) First Pass c) Second Pass d) Coloured Result

While these are the most basic solutions for the problem, a number of authors proposed enhancements to improve their performance, the most prominent ones being the Contour Tracing Labelling by F. Chang et al. [CCL04], a method that uses contour tracing to identify internal and external contours of each component. These contours are then used to label the connected components when the image is traversed horizontally by counting the number of contours passed. The Hybrid Object Labelling presented by Herrero [MH07] presents a method that combines recursive and iterative analysis, finding unlabelled pixels and recursively finding their neighbours until a labelled one is found, then proceeding to the assignment in the adjacent rows.

```
Input: Set of points in the plane
Output: List of vertices of the convex hull in clockwise order

Sort the points by their x-coordinate, creating a sequence p_{1}, p_{2}, ..., p_{n}

Put point p_{1} and p_{2} in a list called L_{upper}
for i=3 to n
  Append p_{i} to L_{upper}
    While L_{upper} has more than two points and the last three points don't form a
        right turn
      Delete the middle of the last three points from L_{upper}

Put points p_{n} and p_{n-1} in a list called L_{lower}
for i=n-2 down to n
  Append p_{i} to L_{lower}
    While L_{lower} has more than two points and the last three points don't form a
        right turn
      Delete the middle of the last three points from L_{lower}

Remove first and last point from L_{lower} to avoid duplication of the points where
    the upper and the lower hull meet
Append L_{lower} and L_{upper} and return that list
```

Listing 2.1: Convex Hull calculation

The implementation of OpenCV is based on the work of Grana et. al [GBC10]. It uses a new concept in this area that the authors describe as a single entry decision table and enhancing them to create the OR-decision tables that allow multiple equivalent decisions for the same conditions. The most convenient decision is selected by creating a decision tree, branching it whenever multiple decisions are possible. To improve the method performance the access to the image is made block-wise instead of the more usual pixel-wise manner. This is the chosen method for implementation in OpenCV due to it's performance compared to other state of the art algorithms.

It is interesting to note the YACCLAB (Yet Another Connected Components Labelling Benchmark) [GBBV16] project, a benchmark platform that allows researchers to compare the performance of their connected components labelling methods to the methods of other researchers. The novelty of this platform is the fact that it uses the authors implementations instead of their own, allowing the algorithms to perform exactly as intended, with all the tweaks and tricks.

2.4.2 Convex Hull

A convex hull is the smallest convex shape that encompasses all the points of the figure. This problem is usually solved in two or three dimensions, although some solutions can solve it for higher number of dimensions [Cha93]. In the field of computer vision however only the second and third dimension are needed.

An algorithm to calculate a convex hull from a set of points [dB08] can be seen in Listing 2.1.

2.5 Object Classification

Some state-of-the-art approaches to the problem of object classification are explored in this section.

2.5.1 Bounding Box Ratio Method

To differentiate between pedestrians and vehicles Jiang Qianyin et al. [QGJX15] provided a method based on the analysis of the bounding box of each blob obtained by the background subtraction method. They propose to compute the ratio of width and height of the rectangle and compare it to pre-calculated tabulated values. Using this approach they are able to distinguish pedestrians from small and large vehicles.

2.5.2 Feature Clustering Method

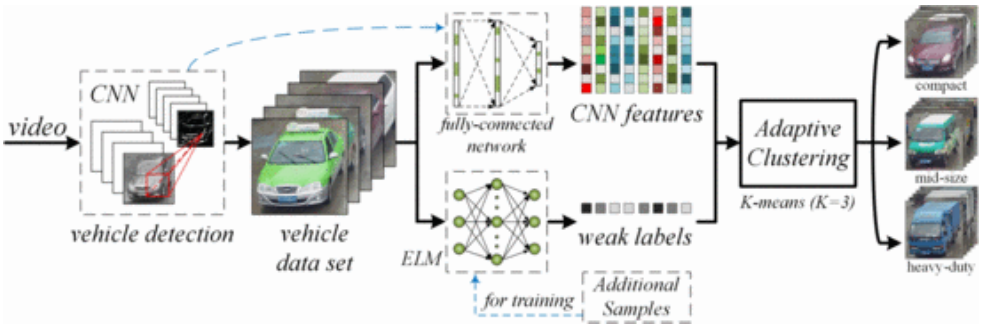


Figure 2.11: Overview of the method as in [WLGC16]

The method proposed by Shu Wang et al. in [WLGC16] categorizes three types of vehicles, compact, mid-sized and heavy-duty. It is described in some detail in figure 2.11 It works in four steps which will now be described.

In the first part a region of interest on a certain video is set manually where a pre-trained Fast Region-based Convolutional Neural Network is used to detect a vehicle in the image.

The second part is the feature extraction, and it performs it in two different forms. One of them uses a Deep Network to extract vehicle features and the other uses a pre-trained Extreme Learning Machine from which the authors obtain a weak label. This ELM is trained with cropped vehicle images and outputs in three different nodes representing the three different vehicle types. Both feature descriptors are fused together using a queue model which can be weighted according to specification.

The algorithm then takes these features and performs k-clustering on them, grouping them in three different clusters representing, once again, the three different vehicle types being analyzed.

After training the system we can use it to classify new samples, measuring the distance of the fused feature to the center of the clusters and choosing the one closer to the new point.

2.5.3 Artificial Neural Network with Histogram of Gradients

In their 2015 work [BG15] Sakan et al. use an ANN with HOG together with an analysis of the geometric features of to achieve better accuracy than the most used methods. A Histogram of Gradients is constructed by computing the gradients of small blocks of the image and grouping them into bins of 20 degrees each based on their orientation. Besides calculating the HOG, the authors also compute the length to height ratio of the bounding box of the vehicle as well as the ratio between the perimeter and the area. The ANN has as input the HOG and the geometric ratios calculated before and is trained with the backpropagation algorithm.

2.6 Fuzzy Sets

Fuzzy sets were introduced by L.A. Zadeh [Zad65] and Dieter Klaua [Kla67] in the mid 60's. Although the work from Klaua is written in german, a very detailed analysis of his work can be found in a paper by Siegfried Gottwald [Got10].

Fuzzy sets are a class of sets where instead of each member belonging to a class like in regular sets, they have a degree of membership to each class ranging from zero to one. This degree is determined by the membership or characteristic function of the class it represents, and can also be seen as a truth value in certain applications.

Since their formulation, the use of these sets as spread to multiple areas: text processing [CBK00] where they are used to model linguistic modifiers taking into account relationships between objects; vehicle safety [DSM12] in which they are used to build a fuzzy deterministic non controller type system that is able to indicate if a battery is on fire given a broad range of inputs; quality of life analysis [PB12] where they are used to analyse the effect of noise pollution caused by road traffic on the efficiency of human workers.

2.7 Traffic Analysis

The topic of traffic analysis using computer vision has been studied by multiple authors who tackled multiple problems. Since our solution is to be able to run on both highway and urban environments, both areas were studied.

2.7.1 Highway Scenarios

Highway scenes are more easily analysed due to the following attributes contributing to a controlled environment:

- Known object types in the scene, as we only have vehicles to analyse, with no pedestrians or other actors present;
- Cameras positioning is studied towards this type of application, meaning there is little to no occlusion of objects;

- The trajectory of the objects has little to no deviation.

In their work "*Detection and classification of vehicles*" [GMMP02] S. Gupte et al. present a method to replace the magnetic loop detectors with a vision-based video monitoring system. Their approach needs minimum scene-specific knowledge, an advantage that speeds up the set-up process. The vehicle tracking method proposed is based on a state machine where a region can belong to one of five different states:

- Update, when a region from the current frame matches exactly to a region in the previous frame;
- Merge, when multiple regions from the previous frame merge into a single one in the current frame;
- Split, when a single region from the previous frame split into multiple regions in the current frame;
- Disappear, when a region from the previous frame is not matched to any region from the current frame;
- Appear, when a region appears and is not near a recently disappeared region;

Using these five states they are able to track vehicles crossing the scene even if they are partially occluded behind other vehicles or scene features. They also present a classification technique that distinguishes cars from non-cars (vans, SUVs, semis and buses) based on the size of the region using a fix threshold value. This was able to achieve a 70% classification success rate in a 20 minutes video.

Goo Jun et al. [JAG08] present a method for segmenting different vehicles inside the same region retrieved by the background subtracter. To do so they calculate a motion vector for each vehicle by tracking feature points across multiple frames and then clustering them based on their velocity.

2.7.2 Urban Scenarios

Urban scenes are harder to analyse due to the following attributes:

- Unknown object types in the scene, making it difficult to segment and classify them;
- Cameras are positioned in low locations, creating a lot of occlusion between objects and other objects, and objects and features in the scene;
- Unexpected trajectories of the objects present in the scene;
- Objects may stop at any point and be classified as background by the segmentation process.

Buch et. al [BOV08] present a method to detect and classify vehicles in urban scenes that use 3D models whose projections are compared to the segmented regions. This method has a reported accuracy of 100% under sunny weather but this value declines when faced with foggy or rainy weather.

Hashmi et. al [HK12] propose an approach to gather statistics of intersection usage, analysing how many vehicles pass through the scene and also from where they come and where they exit. This evaluation is only possible using video, as the traditional methods of counting vehicles cannot establish a relationship between detections at different points of the intersection. This paper however can only deal with free-flowing traffic and has problems with stopped vehicles.

2.8 Summary

This chapter covered the basics of the field of computer vision and its multiple usages, reviewed basic concepts of image treatment, focusing on the ones that would be used during the development of the solution. When reading the work related to traffic analysis we detected a largely unsolved issue, related to the segmentation of stopped vehicles in urban scenes. This review was written taking into consideration that some of the readers may not be familiarized with the area, and thus contains overviews of some basic methodologies that are common in computer vision.

Chapter 3

Methodological Approach

As previously stated, the work done for this project consists of a module for the "Video Server" being developed by LIACC, that will perform all the required analytics of the video received. This chapter provides the details of the implementation of this module, developed during the dissertation semester between February and June 2017. Tasks performed involved designing an appropriate architecture, able to scale with the project, treating the images received from the video streams to detect events and extract information.

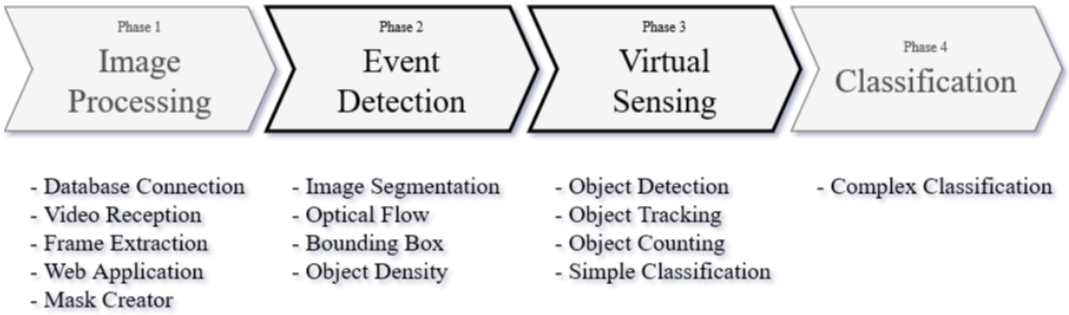


Figure 3.1: Project Planning

It is expected that this chapter provides some insight on how the theoretical backgrounds presented before were used during the project development, namely phase 2 and 3 of the planning presented in 3.1. Phase 2 focused on the detection of relevant events to the traffic controller such as suspicious approximation to the camera, intrusion in prohibited areas, fallen objects on the road and wrong way travelling, while phase 3 aimed to count and classify vehicles in both urban and high-way scenarios.

3.1 Chosen Technology

In order to build this project we needed both a library of already implemented computer vision algorithms as well as a simple way to retrieve frames from both video streams and files. This section describes what were the chosen technologies including a brief description and why it was chosen.

3.1.1 OpenCV

OpenCV is a library composed of implementations of useful computer vision algorithms implementation, widely used across the industry and academy. It has interfaces for multiple programming languages, like C++, Java and Python, but is natively written in C++ in order to take advantage of low level performance enhancements, as performance is an important factor in real time computer vision applications [Ope17a].

The library contains over 2500 algorithms ranging from the more basic image processing, such as filtering, morphology operators and geometric transformations, to more complex ones that are able to compare images, track features, follow camera movements and recognize faces, among others. Along with the image processing capabilities, OpenCV also ships with interfaces to stereo cameras such as Kinect that allow users to retrieve a cloud of 3D points and a depth map from the captured image. This was the chosen library as there existed already previous work at LIACC using it, which could be leveraged for this project.

3.1.2 JavaCV

JavaCV is a wrapper for OpenCV written in Java that works on top of the JavaCPP Presets, a project that provides Java interfaces for commonly used C++ libraries, such as OpenCV and FFmpeg, the ones we are using, as well as CUDA, ARToolKitPlus and others. It provides access to all the functionalities of OpenCV inside a Java environment, and was the chosen solution as there was already experience inside LIACC working with this technology.

Even though the code is written in Java and runs inside a Java Virtual Machine, the code from OpenCV is compiled from C/C++ and the memory of the objects created there is allocated in a separate thread. This made it impossible to rely on the garbage collector to do the memory management, and necessary to manually delete the native objects. Failure to address this issue causes the system to run out of memory and a subsequent program crash.

3.1.3 FFmpeg

FFmpeg is a framework that performs encoding, decoding, transcoding, streaming and filter operations in all the well known video formats, across a large number of platforms. It is used in the "Video Server" to read streams from any format and restream them all to the same format. In the Analytic module it is used to grab the frames from both the video files and the streams using the same code, as shown in 3.1.

```
// videopath can be either a stream path or a file path
FFmpegFrameGrabber _frameGrabber = FFmpegFrameGrabber.createDefault(videopath);
Frame currentFrame = null;

while( (currentFrame = _frameGrabber.grabImage()) != null) {
  // Use grabbed frame
}
```

Listing 3.1: FFmpegFrameGrabber usage example

3.2 Techniques

The following section presents a list of the techniques applied during the project, as they were described in Chapter 2, now diving into more technical details. Following this section a reader can implement a similar solution to the one presented in this document.

3.2.1 Segmentation

This section describes how the segmentation of the received images is performed, and how it returns a foreground mask representing the moving areas of the scene.

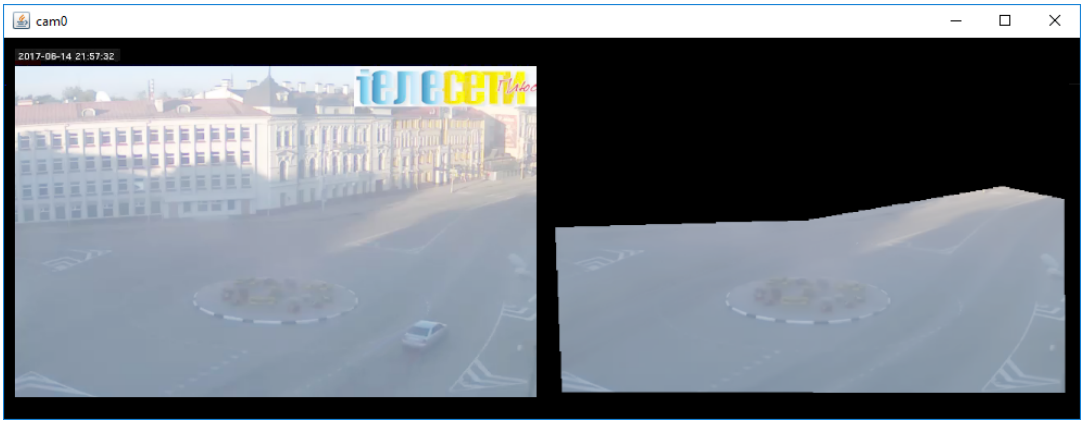


Figure 3.2: Background Subtraction - Background Model

In Figure 3.2 we can see the original frame on the left and the calculated background model of the scene on the right. To achieve this result, the first step is to mask the obtained frame in order to prevent uninteresting regions of the image from being processed by the Background Subtracter. The masking process consists in placing a binary image, called a mask, over the original image and removing all the information where the mask has false values.

The application of the mask solves an issue where moving or changing regions of the image outside our area of interest would create unwanted artefacts, for example moving trees due to the wind blowing, or the issue that occurred in this scene, where a car passing would be reflected in the windows of the building.

The next step of the Segmentation process is to feed the masked frames into a Background Subtraction algorithm that will use them to update its internal representation of the scene. This project uses the OpenCV implementation of the Mixture of Gaussians algorithm that allows the user to tune:

- The number of past frames considered on the background calculation
- The threshold value from which a pixel is considered to be foreground, compared to the difference between its current value and the one from the background model
- The learning rate of the algorithm, how much each new frame influences the model

After updating the background model we can retrieve from the Background Subtracter its foreground mask, a rough representation that is calculated by subtracting the background model from the current image and thresholding it, thus returning only the pixels where the difference is significant enough.

Figure 3.3: Background Subtraction - Foreground Mask

As we can see in the middle image of Figure 3.3 the foreground mask from the Background Subtracter can have a lot of noise due to lighting conditions. To solve this issue two morphology filters are applied to the image: a squared erosion filter to remove the small speckles that appear in the mask; followed by a larger circular dilation filter to consolidate the positive regions of the mask, as the wind shield and windows of the vehicles are usually detected as background due to their dark colour and/or reflection of the environment.

3.2.2 Object Detection

This section describes how the information about the objects' size and position are retrieved from the binary mask returned from the segmentation process.

OpenCV provides BlobDetector [Ope17b], a family of functions that retrieve information of blobs in an image. It works by thresholding the input into multiple binary images using a range of threshold values, grouping the connected white pixels at each one of these images in binary blobs and then merging those that are close enough to each other into larger blobs. This method allows users to filter returned blobs based on 3 properties:

- Area - The area of the region
- Circularity - Ratio between the area of the smallest involving circle and the area of the region
- Convexity - Ratio between the are of the convex hull of the region and its area
- Inertia - Elongation of the region (0 for lines, 1 for circles)

This however does not work as intended for our input, an already binarized image where we just need to group the connected pixels. For this the project uses the OpenCV function *connectedComponentsWithStats*, that retrieves the groups of connected pixels in a binary image along with their area, width, height and both the leftmost and topmost coordinate of the group's bounding box. From this data we can create an ImageObject instance that will represent a moving object in the scene, using the regions with a size above a given threshold, in order to prevent detection of small objects or noise that got through the filtering process.

3.2.3 Object Tracking

In order for the solution to correctly analyse the objects of the scene it needs to track them during their lifetime, in other words, establish a relation between the objects identified on a frame with the objects identified in the next one. This is done by iterating through each one of the new objects and finding which existing object is closer to him. If the distance between them is smaller than the size of the existing object then they are matched.

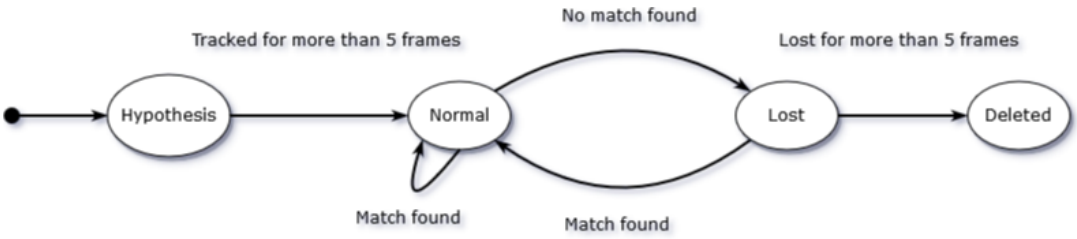


Figure 3.4: Object Life Cycle State Machine

To enhance this mechanism a sub-set of the state machine presented by Jodoin et. al [JBS14] which allows us to model the life cycle of an object. As seen in Figure 3.4 an object starts as an Hypothesis, and cannot be used for analytics while in this state. After being tracked for 5 frames it then changes its state to Normal, where it stays until no match can be found, either from leaving the scene or due to problems with the input frame. An object in this state remains for a maximum of 5 frames, after which it is deleted or until a new object is found in the vicinities of it's last known position.

This technique relies on objects appearing close to their previous position in the next scene, and because the project was designed to run in high-way scenarios, where vehicles travel at high speeds, there was the need to improve it. The chosen solution was to follow a method used by Chris Dahms in his vehicle counting software [Dah17] that predicts an object next position by

calculating an weighted average of the previous inter-position deltas and summing it to its current position as shown in 3.1.

$$NextPos = CurrentPos + \sum_{i=1}^{5} (PosHist(i) - PosHist(i-1)) * (5-i) \tag{3.1}$$

In figure 3.5 we can see the result of the tracking process on the right most image. Each detected object is circled in a colour representing its current state in the state machine, blue for normal and red for lost, and shows the trail represents its path across the scene.

Figure 3.5: Object Tracking

3.2.4 Object Counting

This section will provide an overview of the environment created to support the development of the project, a flexible multi feed visualizer as seen in Figure 3.6. The top-left image shows the tracking process with the green zones indicating the virtual counting sensors. When a vehicle enters the sensor it is counted and classified. The middle top image shows the background model as calculated via the background subtracter explained in Section 3.2.1. The top right image and the bottom left one show the foreground mask before and after being processed by the morphological operators to remove the noise.

3.2.5 Object Classification

In order to classify the objects present in the scene into light or heavy vehicles a fuzzy set is used. This set is calculated at run time based on a mask drawn by the user via the web interface that roughly approximates the size of a light vehicle. The area of that mask (A_{Light}) is used as a base value to create the fuzzy set shown in figure 3.7. This set has 2 series, one for light vehicles, drawn in blue, and one for heavy vehicles, drawn in red.

The blue series peaks at A_{Light}, where we have 100% certainty that a matched object is a light vehicle. The point immediate points are at 2/3*A_{Light}, where the trust is 80% and at 4/3*A_{Light} where it drops to 60%. Any object with area below 1/2*A_{Light} or above 2*A_{Light} are considered to have 0% chance to be light vehicles.

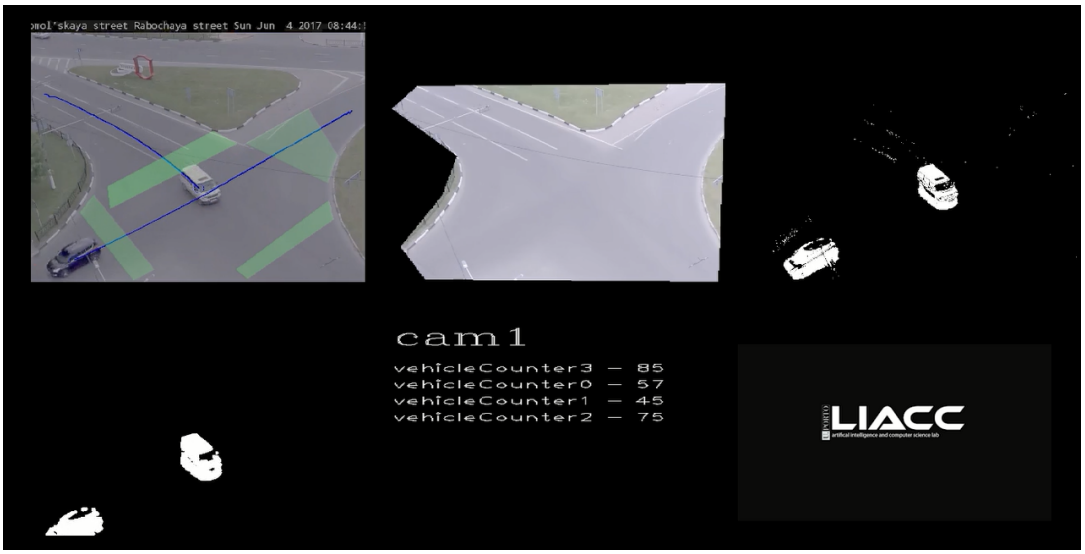


Figure 3.6: Multi-Feed Visualizer

The red series peaks at 2*A_{Light}, and any object whose area is larger than this value is considered to be an heavy vehicle with 100% confidence. At the same time, any object whose area is smaller than 4/3*A_{Light} is never considered to be an heavy vehicle, from where the certainty rises to 80% at 5/3*A_{Light}.

When an object is counted in one of the virtual sensors its area is passed to this classifier where an interpolation is calculated for each series, returning the certainty with which it belongs to each one of the classes. Using these values the process can distinguish classifications with certainty below and above a user specified threshold to be treated separately.

3.2.6 Feature Point Tracking

The need to track feature points stems from the fact that without them we rely wholly on the background subtraction to process the image. Using feature points we can extract and track details of the vehicles that would otherwise be lost due to the process only working with a binary mask.

We can use these points to distinguish vehicles grouped in a single binary region, as they are characterized by these features. This way, even if a vehicle stops for a long time and starts to be considered background we can use its tracked points to maintain the position of these stopped vehicles as they regain motion.

3.3 Summary

Although dealing with a framework with no proper documentation was a difficult challenge to overcome, the performance obtained from the native code was essential for the final result, as the software is now able to track objects in a busy scene faster than the camera's capture rate, allowing us to process a video file in a shorter period of time than it's length.

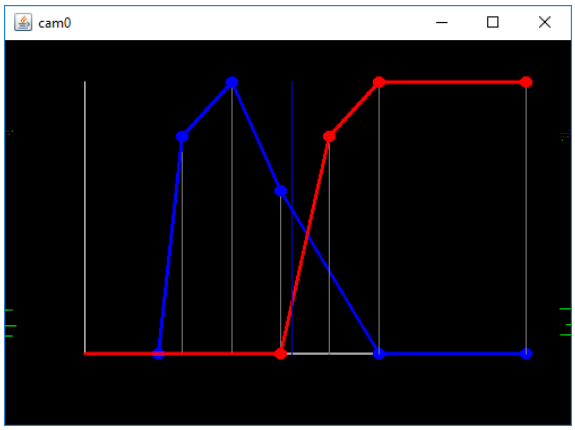


Figure 3.7: Object Classifier

Chapter 4

Implementation and Results

This section describes the architecture of the implemented solution as well as the results obtained during the testing phase of the project.

4.1 Architecture

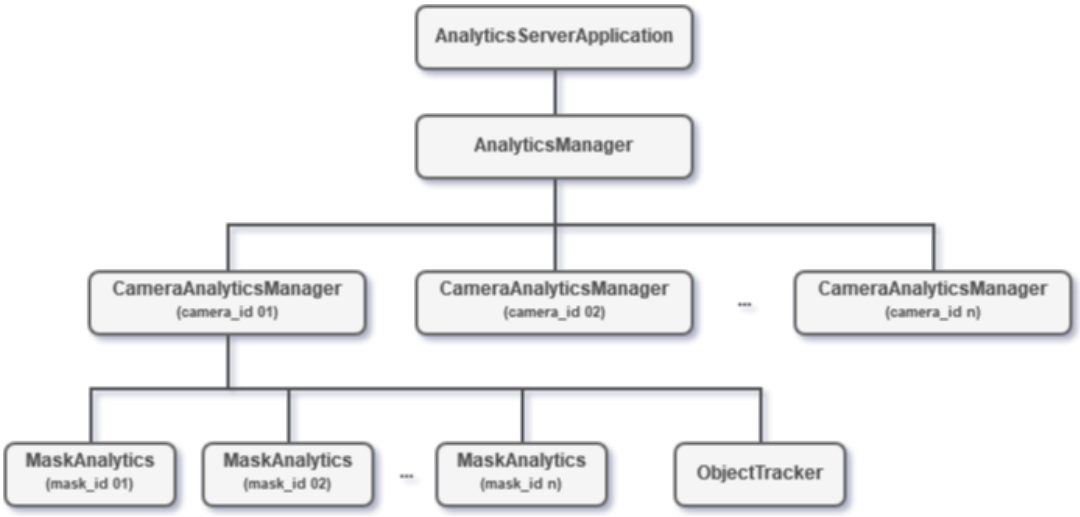


Figure 4.1: Manager Architecture

The Analytics module was designed to comply both with the specifications imposed by our partner and with the characteristics of the modules already developed. The requirements are the following:

- Run uninterruptedly waiting for requests to be made;
- Allow for video analysis with no significant delay;

- Process a large number of video inputs at the same time;
- Receive input from streams or files;
- Specify which analysis are to be run on a specific video;
- Use free-to-use or open-source tools;
- Allow for deployment scalability.

The application main thread runs an instance of the AnalyticsManager class that implements the Java Runnable interface. This thread will launch one CameraAnalyticsManager instance for each video to be analysed, be it from stream or from file, and keep track of each worker status. When a camera is disconnected from the server the corresponding thread is terminated. Likewise, when dealing with video from files, the thread is terminated when the all the analytics processors linked to that video are finished. The AnalyticsManager then enters a loop, pooling the database at a fixed interval, querying for cameras added to the system. When one is found, a new CameraAnalyticsManager is launched, linked to that camera. The process can be seen in detail in Algorithm 4.1.

```
1  string query = "SELECT * FROM camera WHERE do_alarms OR do_virtual_sensors";
2  list<Camera> camerasWithAnalytics = database.query(query);
3
4  foreach camera in camerasWithAnalytics
5    startNewCameraAnalyticsManager(camera);
6  end
7
8  while(!stopThread)
9    newCamerasWithAnalytics = database.query(query);
10   foreach camera in camerasWithAnalytics
11     if(!camera.beingAnalyzed)
12       startNewCameraAnalyticsManager(camera);
13     end
14   end
15   wait(5000);
16 end
```

Algorithm 4.1: AnalyticsManager

The CameraAnalyticsManager is then responsible to query the database in order to find out what are the analytics the user wants to retrieve from the video through information that is stored in the *camera* table of the database. As of now there are three types of analytics that can be run over each input: Vehicle Tracking, Wrong-Way Travelling Detection and Suspicious Camera Approximation. Depending on the result of the query, a new thread is launched to perform each of the required analytics.

This process then starts a frame grabber that will convert each frame of the video into its representation in OpenCV and feed it to each Analyser through a queue. This enables each one

of these workers to run at his own pace and if one of them runs slower than the pace at which the frame grabber reads the images, it will simply queue them up and not slow down the remaining workers. The drawback of this solution is that it is theoretically possible to run out of memory to keep these frames, although this limit was not reached during the testing phase. The process can be perceived in more detail through the analysis of Algorithm 4.2.

```
1
2  cameraAnalytics = getAnalyticsFromDb(camera);
3
4  if (cameraAnalytics.AlarmAnalytics)
5    if(cameraAnalytics.WrongWay)
6      launchWrongWayThread(camera)
7    end
8    if(cameraAnalytics.CameraAproximation)
9      launchCameraAproximationThread(camera)
10   end
11 end
12
13 if (cameraAnalytics.VehicleCounting)
14   launchObjectTrackerThread(camera)
15 end
16
17 while(!Thread.interrupted && (currentFrame = frameGrabber.grabImage())!=null)
18   foreach createdThread
19     createdThread.frameQueue.put(currentFrame)
20   end
21 end
```

Algorithm 4.2: CameraAnalyticsManager

4.2 Image Processors

In order to streamline the use of some of the OpenCV functionalities a series of helper classes were created to help adjust the parameters of the algorithms during the development phase. These processors were implemented as subclasses of ImageProcessor, a superclass described in Listing 4.3.

The most relevant method of the described class is *process(Mat original)*, the one that will be called in the code to use the processor. Each implementation of this abstract class will then contain an unique *imageProcessorType* that will allow us to distinguish the objects from each of them.

The *ImageProcessor* subclasses are listed in Table 4.1, along with their purpose and the parameters that can be tweaked.

An addition to this functionality was made by introducing the concept of Image Processing Pipelines. This is a structure that groups multiple Image Processors into a single object, enabling us to run an image over all of the processors in a single call. The pipeline is created as an empty list and then the previously created processors are queued in the order they will be run.

```
1
2  public abstract class ImageProcessor {
3      protected final String imageProcessorType;
4
5      protected ImageProcessor(String imageProcessorType) {
6          this.imageProcessorType = imageProcessorType;
7      }
8
9      public abstract void process(Mat original);
10
11     public String getImageProcessorType() {
12         return imageProcessorType;
13     }
14 }
```

Listing 4.3: ImageProcessor class

Table 4.1: Image Processors

ImageProcessor	Purpose	Parameters
BackgroundSubtractor	Perform background subtraction over a sequence of frames	Learning rate of the background model, number of frames that are considered in the calculation, threshold for foreground consideration
ColorChanger	Change colour scheme from black and white to colour and vice-versa	Direction of conversion
EdgeDetector	Detect edges in an image	First and Second threshold necessary for the Canny Operator
ImageResize	Change the size of an image	New size for the image
ImageSmoother	Perform a gaussian blurring of the image	Size of the Gaussian filter
MaskProcessor	Mask the input image	Binary image containing the mask
Morphology	Apply a morphological operator to the input binary image	Which of the morphological operators to be run, the shape and size of the kernel
ShapePlacer	Draw a rectangle, circle or convex polygon in an image	Which shape to draw, color and where to draw it
TextWriter	Write text in an image	Text to be written, font, size, color and lower-left coordinate
Thresholder	Perform a thresholding operation over the image	Value for the threshold operation

4.3 Multi-Feed Visualizer - Video Wall

The need for a multi feed visualizer arose during the development phase, as having the program open a new window for each feed that needed to be visualized was slowing down the development environment. The implemented solution overcomes this issue as it only needs a single window

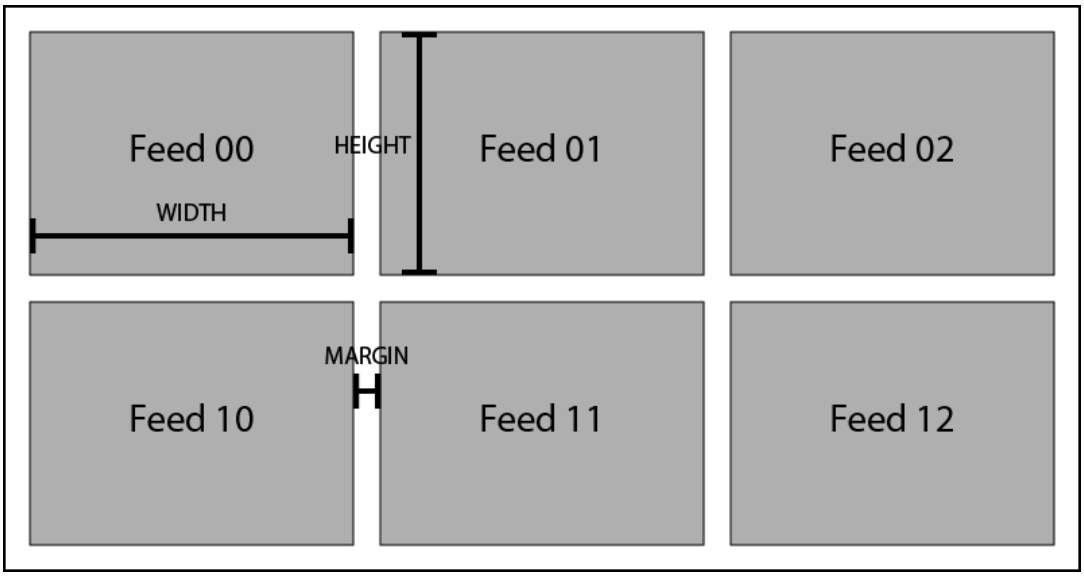


Figure 4.2: Multi-Feed Visualizer 2x3 Example

and consequently, a single draw call.

The Multi-Feed Visualizer is implemented in the *VideoWall* class, shown in Listing 4.4. It takes as input the desired individual feed width and height output, the margin to be placed between the various feeds and the number of rows of columns that compose the feed visualization matrix. An example of the resulting structure can be seen in Figure 4.2. As for the implementation of the class, it can be seen in Listing 4.4 and will now be detailed.

The VideoWall class relies heavily on the OpenCV concept of Region of Interest, a feature that allows for the selection of a rectangular area of an image and to display another image inside said rectangle. When an object of the VideoWall class is created, an empty image is created with the dimensions calculated as shown in Equation 4.1 and 4.2.

$$Width = nCols * videoWidth + (nCols + 1) * margin \tag{4.1}$$

$$Height = nRows + videoHeight + (nRows + 1) * margin \tag{4.2}$$

On this image we then calculate the regions of interest equal to the number of cells in the matrix at the specified positions. The *show* method then iterates through these regions and places a resized version of the input feed in each of the rectangles. These images also need to be converted to RGB colour space, as this is the one used by the output image.

```
1  public class VideoWall {
2      // Configurable parameters
3      private int videoWidth, videoHeight, margin, nRows, nCols;
4      // Computed
5      private int totalWidth, totalHeight;
6
7      private ImageResize imageResize;
8      private ColorChanger colorChanger;
9      private Mat wall;
10     private ArrayList<Mat> ROIs;
11
12     public VideoWall(int videoWidth, int videoHeight, int margin, int nRows, int
           nCols) {
13         this.videoWidth = videoWidth;
14         this.videoHeight = videoHeight;
15         this.margin = margin;
16         this.nRows = nRows;
17         this.nCols = nCols;
18
19         imageResize = new ImageResize(videoWidth,videoHeight);
20         colorChanger = new ColorChanger(ColorChanger.GRAYSCALE_TO_BGR);
21
22         totalWidth = nCols * videoWidth + (nCols + 1) * margin;
23         totalHeight = nRows * videoHeight + (nCols + 1) * margin;
24         wall = new Mat(totalHeight,totalWidth, opencv_core.CV_8UC3);
25
26         ROIs = new ArrayList<>();
27         generateROIs();
28     }
29
30     public Mat show (Mat... frames) {
31         if (frames.length > nRows*nCols) {
32             LOGGER.error("Exceeded number of streams");
33             return wall;
34         }
35         for (int frame = 0; frame < frames.length; frame++) {
36             Mat frameCopy = frames[frame].clone();
37             imageResize.process(frameCopy);
38             if(frameCopy.channels() != 3) {
39                 colorChanger.process(frameCopy);
40                 frameCopy.copyTo(ROIs.get(frame));
41             }
42             else {
43                 imageResize.process(frameCopy);
44                 frameCopy.copyTo(ROIs.get(frame));
45             }
46             frameCopy.release();
47         }
48         return wall;
49     }
50     // Generate the regions of interest to be used in show()
51     private void generateROIs ();
52 }
```

Listing 4.4: Video Wall class

```
1  // Create BackgroundSubtractor processor
2  BackgroundSubtractor bgSub = new BackgroundSubtractor(10000,32);
3
4  // Create foregroundMaskCreator pipeline
5  ImageProcessingPipeline foregroundMaskCreator = new ImageProcessingPipeline()
6                  .addThresholder(30, opencv_imgproc.THRESH_BINARY)
7                  .addMorphology(Morphology.EROSION,3,opencv_imgproc.CV_SHAPE_RECT)
8                  .addMorphology(Morphology.DILATION,8,opencv_imgproc.
                       CV_SHAPE_ELLIPSE);
9
10 (...)
11
12 private void SegmentImage() {
13   bgSub.process(maskedImage);
14   foregroundMask = bgSub.getForegroundMask();
15   foregroundMaskCreator.applyImageProcessors(foregroundMask);
16 }
```

Listing 4.5: ImageProcessor class

4.4 Image Segmentation

This section will provide insight on how the image segmentation portion of the implementation was approached, from a technical point of view. We will follow the approach detailed in Section 3.2.1, using the Background Subtracter algorithm and cleaning the resulting foreground mask with a collection of morphological operators. The process is shown in Java code in Listing 4.5.

As described above, before the image is processed by the background subtracter algorithm, uninteresting regions present are removed. This is done using a MaskProcessor processor using as mask a combination of all the lane masks created by the user. By adding this step in the algorithm we remove the possibility of detecting moving objects that are not interesting for the application, such as clouds or reflection of vehicles in windows of buildings.

To segment the resulting image a BackgroundProcessor processor is used. This processor uses the *BackgroundSubtractorMOG2* Background Subtracter implementation from OpenCV, a mixture of gaussians based approach. At each iteration the new frame is passed to the processor which updates the background model and consequently the foreground mask.

In order to clear the noise and enhance the regions on the foreground mask, an Image Processing Pipeline is created, the *foregroundMaskCreator*. This pipeline contains three Image Processors: a Thresholder with a threshold value of 30, a Morphology set to erosion with a three by three rectangular kernel and another Morphology set to dilation with an eight by eight elliptical kernel. The application of this pipeline on the foreground mask removes any pixels not different enough from the background, removes small grain noise and consolidates the regions by dilating them.

4.5 Object Detection

The process of detecting objects in a scene from the foreground mask is now detailed. For this task we will follow the approach presented in Subsection 3.2.2, running a connected component labelling algorithm over the treated foreground mask. The results of this operation are then analysed and the objects are created.

The objects in the scene are modelled according to the *ImageObject* class, shown in Listing 4.6. In this Section we will review the process of the detection, while Section 4.6 will describe the tracking process across frames.

The detection from the foreground mask is done using the OpenCV implementation of the connected components labelling, *connectedComponentsWithStats*. This function takes as input the binary image to be labelled, the connectivity to be considered, four or eight, and references to where the results will be written. These include a matrix for the centroids of the regions, one for the labels and another one for the statistics of the regions: area, width, height and left and top coordinates.

This information is then used to create an ImageObject object, where the *centroid* is the centre of the region, the *imageSize* is the area of the region and the size is the diagonal length of the rectangle encompassing the region, calculated from the top-left coordinate and the width and height. Not all regions form a new object however, as regions with area below a given threshold are excluded from the process. This ensures that noise that was not removed during the morphological operators application does not form non existent objects.

Although this is not used for applicational purposes, the developed process also tracks feature points on objects. This can be used to further enhance the segmentation process, distinguishing between multiple objects in a single foreground region. To track feature points only on vehicles, the image is masked with the foreground mask and then the OpenCV method *goodFeaturesToTrack* is called. This method takes as input a greyscale image, a list of feature points detected in the previous frame of the video, maximum number of corners to be detected, a quality level (each feature point has a "strength" value associated, this quality level is a percentage that indicates that the algorithm should only return points whose "strength" is above that percentage of the "strongest" point) and the minimum distance between points. The retrieved points are then associated with the closest object.

4.6 Object Tracking and Counting

This section will describe how the objects are tracked across the scene, as well as how they are counted in multiple virtual sensors. The process involves comparing the list of objects present in consecutive frames and establishing a relation between them.

From the previous step, described in section 4.5, we get a list of objects currently present in the scene, their position and size. In the first frame of the image, these objects are added to the list of scene objects with their current properties. In the following frames, a map structure is created to

```
public class ImageObject {

    public static class State{}

    public static class FeaturePoint {
        public boolean updated = true;
        public Point position;
        public List<Point> positionHistory;

        public FeaturePoint(Point position);

        public void Update(Point newPos);
    }
    // List of possible object states
    public static final State Hypothesis;
    public static final State Normal;
    public static final State Lost;
    public static final State Deleted;

    public State currentState;
    // Object properties
    public Point centroid;
    public Point predictedCentroid;
    public float size;
    public float imageSize;
    public List<Point> centroidHistory;
    public Rect boundingRect;
    public boolean counted = false;
    public List<Mask> countedOn = new ArrayList<>();
    // Number of frames required for the object to be considered lost
    private int nLostFrames = 5;

    public List<FeaturePoint> features = new ArrayList<>();
    // Class constructor
    ImageObject(Point centroid, float size, float imageSize);
    // Method to update the features of an object
    public void UpdateFeaturePoints(List<Pair<Point, Point>> pairs);
    // Helper method to calculate the euclidean distance between two points
    static double distanceBetweenPoints(Point point1, Point point2);
    // Method to update the position, size and state of an object
    public void UpdateObject(ImageObject object);
    // Draw an object representation in the input image
    public void DrawObject(Mat image);
    // Mark an object as counted on the specified virtual sensor
    public void setCounted(Mask mask);
    // Treat the event of an object not being found
    public void notFoundEvent();
    // Predicts the position of the object in the next frame
    public void predictNextPos();
    // Returns the position of the object in the N past frame
    private Point getNLastPos(int index);
}
```

Listing 4.6: ImageObject class

relate a previously existing object in the scene to the ones present in the new one. The relationship structure is populated with the new objects by calculating the euclidean distance of an object to all the other objects. The new object is then added to the list related to the object with which the distance is smaller.

Using the list related to each of the previously present objects we then update the position of the existing objects with the one of closest newly detected object, as well as updating the size. When an history of positions is kept through multiple frames, the object can be tracked across the scene. The evolution of the object states is done according to the state machine explained in Section 4.6. The events that update the state of the object are retrieved from the relationship information. If a previously existing object has no objects in the vicinity in the new frame, it receives a "No match found" event, otherwise, it receives a "Match found" one. With these two events we can update the state according to the state machine presented in Figure 3.4.

The tracking process implementation can be analysed by the reader in Listing 4.7, where the code that implements the described functionalities is present. This code is run once per frame, using the image retrieved from the frame queue.

In order to count the objects, their position is used to retrieve the pixel colour from the binary images that contain the virtual sensors zones. When this operation returns white, it means that the object is over a counting zone, and as such it is marked as being counted in that zone already. This approach guarantees that an object is not counted multiple times by the same sensor.

4.7 Classification

The object classification is performed every time it is counted in a virtual sensor. This is due to the fact that the system used needs an user input value that estimates the expected area of a region containing a single Light vehicle. This value changes according to the distance from the sensor to the camera, as well as viewing perspective, and as such we need one classifier for each classifying zone.

The classifier fuzzy set is described in detail in Section 3.2.5, and we will now only discuss the implementation of the theoretical aspects already presented. The *ObjectClassifier* class is presented in Listing 4.8 and will now be detailed. It contains two inner classes: *Variable* and *FuzzySet*. The *FuzzySet* class contains all the logic for the fuzzy set, while the *Variable* class represents a class of the set. Each class is composed of a series of points that determine what the degree of belonging is at that point.

When an object area is sent to be classified, an interpolation is made between the closest neighbouring points of each class. The values are then compared and the largest one is returned. If this value is below the *certainThreshold* the result of the operation is an "Unknown" classification, otherwise it is the name of the class which returned the largest value for the given input.

```
boolean[] matched = new boolean[currentObjects.size()];

// Map each new object to the previous ones
Map<ImageObject,List<ImageObject>> relations = new HashMap<>();
for (ImageObject prevObj:
        currentObjects) {
    relations.put(prevObj,new ArrayList<>());
}
List<ImageObject> objectsToAdd = new ArrayList<>();
// For each object detected in this frame
for (ImageObject newObject :
        newObjects) {
    int leastDistIndex = -1;
    double leastDist = Double.MAX_VALUE;

    // Compare distance to each existing point
    for (int existingObjectIndex = 0; existingObjectIndex < currentObjects.size();
        existingObjectIndex++) {
        ImageObject existingObject = currentObjects.get(existingObjectIndex);
        double dist = ImageObject.distanceBetweenPoints(newObject.centroid,
            existingObject.predictedCentroid);
        if (dist < leastDist) {
            leastDist = dist;
            leastDistIndex = existingObjectIndex;
        }
    }
    if (leastDist < newObject.size) {
        currentObjects.get(leastDistIndex).UpdateObject(newObject);
        matched[leastDistIndex] = true;
        relations.get(currentObjects.get(leastDistIndex)).add(newObject);
    } else {
        objectsToAdd.add(newObject);
    }
}

for (int currentObjectIndex = 0; currentObjectIndex < currentObjects.size();
    currentObjectIndex++) {
    if (!matched[currentObjectIndex]) {
        currentObjects.get(currentObjectIndex).notFoundEvent();
    }
}

currentObjects.removeIf(object -> object.currentState == ImageObject.Deleted);
currentObjects.addAll(objectsToAdd);
```

Listing 4.7: Tracking Process

```
1
2  public class ObjectClassifier {
3
4      class Variable {
5          String Name;
6          List<Pair<Float,Float>> BreakPoints;
7          opencv_core.Scalar debugColor;
8          Variable(String name, opencv_core.Scalar debugColor, Pair<Float, Float>...
               breakpoints) {...}
9      }
10
11     class FuzzySet {
12         List<Variable> Variables;
13
14         FuzzySet(Variable... variables) {...}
15         Map<Float,Variable> Evaluate(Float value);
16     }
17
18     private List<CountEvent> countEvents = new ArrayList<>();
19     private float maxValue;
20     public float certainThreshold = .65f;
21     FuzzySet vehicleClassifier;
22     public opencv_core.Mat classifierMat;
23
24     public ObjectClassifier(float vehicleArea) {
25         maxValue = 3*vehicleArea;
26         AnalyticsManager.LOGGER.debug("Creating Classifier with input {}",
               vehicleArea);
27         vehicleClassifier = new FuzzySet(
28                 new Variable("Light",...),
29                 new Variable( "Heavy",...));
30         classifierMat = new opencv_core.Mat(360,480,opencv_core.CV_8UC3);
31         drawClassifier();
32     }
33
34     public Pair<String,Float> getVehicleClass(float value) {
35         Map<Float,Variable> results = vehicleClassifier.Evaluate(value);
36         Float maxVal = 0f;
37         for (Float val :
38                 results.keySet()) {
39             if (val > maxVal)
40                 maxVal = val;
41         }
42         countEvents.add(new CountEvent(value,results.get(maxVal)));
43         if (maxVal > certainThreshold)
44             return new Pair<>(results.get(maxVal).Name,maxVal);
45         else
46             return new Pair<>("Unknown",0f);
47     }
48
49     // Draw the fuzzy set of the classifier
50     private void drawClassifier();
51
52 }
```

Listing 4.8: ObjectClassifier class

4.8 Results

In this section we will present the results achieved by our approach in different settings, which are described below. All the experiments consisted of reading a video stream during a period of time and running our algorithm to perform vehicle counting and classification.

(a) Set-up 1

(b) Set-up 2

(c) Set-up 3

(d) Set-up 4

Figure 4.3: Frames from videos used to test

The tables that list the results have in the first row the true values, as observed in the video and the second row presents the number of detected vehicles by our algorithm. The final row shows the ratio of detected over observed vehicles. In the left column of data are the numbers relating to the Light vehicles, and in the one to the right are the numbers relating to the Heavy vehicles.

4.8.1 Set-up 1

In this set-up the stream was captured in the city of Porto, over the Via Cintura Interna, a major motorway that crosses the urban area. The camera is positioned in an overpass, to the left of the road, meaning that the fast lane is in the foreground. During the recording of the video the traffic intensity was high. The results for this set-up can be seen in Table 4.2.

4.8.2 Set-up 2

In this set-up the stream was once again captured in the city of Porto, over the Via Cintura Interna. The camera is positioned in the same overpass, but this pointed parallel to the road, between the

Table 4.2: Results from Set-up 1

	Light	Heavy
Observed	141	22
Detected	126	20
Ratio	89.3%	90.9%

lanes. From this perspective both lanes are seen as described in set-up 1. During the recording of the video the traffic intensity was also high. The results for this set-up can be seen in Table 4.3.

Table 4.3: Results from Set-up 2

	Light	Heavy
Observed	155	30
Detected	131	28
Ratio	84.5%	93.3%

4.8.3 Set-up 3

In this set-up the stream was also captured in the city of Porto, over the Via Cintura Interna. The camera is positioned in the same overpass, this time to the right of the lane, positioning the slower lanes in the foreground. During the recording of the video the traffic intensity was also high. The results for this set-up can be seen in Table 4.4.

Table 4.4: Results from Set-up 3

	Light	Heavy
Observed	178	28
Detected	176	24
Ratio	98.9%	85.7%

4.8.4 Set-up 4

In this set-up the stream was captured in a non-urban motorway, in the south of Portugal. The camera is positioned very high above the road, on the outside of the lane. In this test we count both incoming and outgoing lanes. However, for these zones, where the objective of the camera is to control the automatic tolling ports, there is one camera for each lane. The traffic intensity during the video is low. The results for this set-up can be seen in Table 4.5.

Table 4.5: Results from Set-up 4 (Incoming and Outgoing Lanes, respectively)

	Light	Heavy	Light	Heavy
Observed	47	5	30	0
Detected	38	5	22	0
Ratio	80.9%	100%	73.3%	-

4.8.5 Discussion

Due to the fact that we use a fuzzy set to classify the vehicles based on their segmented region area, we can know with which degree of certainty the system classifies each vehicle as Light or Heavy. We can use this value to threshold weak classifications and, depending on the application, send the image to a more sophisticated classifier or even to a human operator.

The main drawback of the fuzzy set based classification is its dependency on the object identification process. As our segmentation process returns a binary foreground mask it is impossible to distinguish between multiple vehicles in a single foreground region, and only one object is detected. This object area is then taken into account by the classifier, and in cases where multiple Light vehicles occlude each other, they are classified as a single Heavy vehicle.

The main factors that contribute to the performance of our counting and classification processes are the camera position and the traffic intensity, as seen in the results. In videos one through three the medium to high density of traffic cause multiple occlusions which result in wrong classifications as explained before. In cases where the camera is placed in a high position overlooking the road, such as in video four, the occurrence of vehicle occlusion is reduced.

4.9 Summary

This chapter analysed the architecture of the implementation, reviewing its advantages and shortcomings. The results obtained in the testing phase are presented and discussed, presenting the findings of the experiences.

Chapter 5

Conclusions and Future Work

This chapter presents an overview of the work done, as well as an evaluation of the state of completion achieved during the semester, followed by a list of the contributions and a foresight into the next steps of the project.

5.1 Contributions

The contributions of the dissertation can by split into Technological, Scientific and Applicational contributions.

Technological Contributions During the dissertation work an issue was found when trying to process a frame in multiple threads at the same time. The solution first implemented meant that the frame collector had to wait for each thread to return for it to go fetch the next one, which meant that some of the threads could have potentially long waiting times. The solution now implemented, described in section 4.1, creates a queue in each of the threads to where the frame collector sends the frames and from where the thread processing them reads them. This solution solves the waiting problem by making the threads totally independent from each other and ensures that the whole system performance is not capped if a slower thread is introduced.

Scientific Contributions One of the main contributions of the project is related to the tracking of stopped vehicles found in urban scenarios, as it was one of the gaps found during the literature review. To solve this issue an alteration is proposed in the segmentation process to keep track of the state of stopped vehicles. The first one is a stabilization step in the background subtraction initialization that waits for the background model to be steady. This steadiness is measured by the number of pixels updated in the last frame after the application of the background subtracter. This step ensures that the background is not initialized with stopped vehicles or other actors in the middle of the scene.

Applicational Contributions All the work developed was made taken into account that the application was to be used by our partners at Armis-Group for vehicle counting and classification

in high-way scenarios and as such this was the main focus of the use-cases in which the project was tested. As of the writing of this document an application is being prepared to be tested at Infrastruturas de Portugal with more than 100 camera inputs.

5.2 Publications

During the development of the project a paper was submitted to the "Artificial Transportation Systems and Simulation" workshop in the IEEE International Conference on Intelligent Transportation Systems 2017. The submitted paper is based on the work done during this presentation with emphasis on the ability to feed information onto traffic simulation systems, based on the data gathered from real world.

5.3 Future Work

During the dissertation work a number of challenges appeared that would be interesting to address.

Solve the issue regarding object grouping As of now the segmentation process cannot distinguish multiple objects occluded by one another or linked through a shadow, although there is written work about how to solve this. Implementing such a solution would improve the results of the counting process.

Time counting The counting process can accurately count vehicles but it cannot detect when a vehicle was counted. To do so in videos, a starting time would be required as well as the frame count and rate. In streams the application needs to take into account both the starting time and the stream delay.

Intersection Analysis It would be interesting to follow some of the work regarding intersection statistics and implement a module on top of the Analytics Module that would work specifically for such cases.

5.4 Final Remarks

The work regarding this dissertation can be divided into two main objectives: event detection and vehicle counting. As specified in the beginning of chapter 3, phase two of the project dealt with the event detection portion of the work and phase three with vehicle counting.

During the development of phase two the priority of features shifted to vehicle counting due to a commitment by our partner and as such this phase was aborted to begin the development of phase three. This meant that of the initial list of events to be detected: wrong way driving, suspicious camera approximation, prohibited zone entering and fallen objects only the first three were complete. Regarding phase three, all the points were implemented and the application is now able to successfully count and classify vehicles into the light and heavy categories.

References

[ACM12] ACM. The 2012 ACM Computing Classification System — Association for Computing Machinery, 2012.

[Adm17] Administrator. Intelligent Transport Systems | Armis ITS, 2017.

[AQIS07] A. AbuBaker, R. Qahwaji, S. Ipson, and M. Saleh. One Scan Connected Component Labeling Technique. In *2007 IEEE International Conference on Signal Processing and Communications*, pages 1283–1286, November 2007.

[ARB15] T. Azevedo, R. J. F. Rossetti, and J. G. Barbosa. A state-of-the-art integrated transportation simulation platform. In *2015 International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, pages 340–347, June 2015.

[Aya98] N. Ayache. Medical image analysis a challenge for computer vision research. In *Proceedings. Fourteenth International Conference on Pattern Recognition (Cat. No.98EX170)*, volume 2, pages 1255–1256 vol.2, August 1998.

[BAR15] J. Barros, M. Araujo, and R. J. F. Rossetti. Short-term real-time traffic prediction methods: A survey. In *2015 International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, pages 132–139, June 2015.

[BG15] Saran K. B and Sreelekha G. Traffic video surveillance: Vehicle detection and classification. In *2015 International Conference on Control Communication Computing India (ICCC)*, pages 516–521, November 2015.

[BMCM97] D. Beymer, P. McLauchlan, B. Coifman, and J. Malik. A real-time computer vision system for measuring traffic parameters. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 495–501, June 1997.

[BOV08] N. Buch, J. Orwell, and S. A. Velastin. Detection and classification of vehicles for urban traffic scenes. In *2008 5th International Conference on Visual Information Engineering (VIE 2008)*, pages 182–187, July 2008.

[BP98] Jorge Badenas and Filiberto Pla. Applying computer vision techniques to traffic monitoring tasks. In *Methodology and Tools in Knowledge-Based Systems*, pages 776–785. Springer, Berlin, Heidelberg, June 1998.

REFERENCES

[BT73] T. O. Binford and J. M. Tenenbaum. Computer vision. *Computer*, 6(5):19–24, May 1973.

[BVO11] N. Buch, S. A. Velastin, and J. Orwell. A Review of Computer Vision Techniques for the Analysis of Urban Traffic. *IEEE Transactions on Intelligent Transportation Systems*, 12(3):920–939, September 2011.

[Can86] J. Canny. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, November 1986.

[CBK00] Martine De Cock, Ulrich Bodenhofer, and Etienne E. Kerre. Modelling Linguistic Expressions Using Fuzzy Relations. In *In 'Proceedings of the 6th International Conference on Soft Computing*, pages 353–360, 2000.

[CBMM98] Benjamin Coifman, David Beymer, Philip McLauchlan, and Jitendra Malik. A real-time computer vision system for vehicle tracking and traffic surveillance. *Transportation Research Part C: Emerging Technologies*, 6(4):271–288, August 1998.

[CCL04] Fu Chang, Chun-Jen Chen, and Chi-Jen Lu. A linear-time component-labeling algorithm using contour tracing technique. *Computer Vision and Image Understanding*, 93(2):206–220, February 2004.

[CGPP00] R. Cucchiara, C. Grana, M. Piccardi, and A. Prati. Statistic and knowledge-based moving object detection in traffic scenes. In *ITSC2000. 2000 IEEE Intelligent Transportation Systems. Proceedings (Cat. No.00TH8493)*, pages 27–32, 2000.

[CGPP03] R. Cucchiara, C. Grana, M. Piccardi, and A. Prati. Detecting moving objects, ghosts, and shadows in video streams. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(10):1337–1342, October 2003.

[Cha93] Bernard Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete & Computational Geometry*, 10(4):377–409, December 1993.

[Dah17] Chris Dahms. OpenCV_3_Car_Counting_Cpp, May 2017.

[dB08] Mark de Berg. *Computational Geometry: Algorithms and Applications*. Springer Science & Business Media, March 2008. Google-Books-ID: tkyG8W2163YC.

[Dou92] Edward R. Dougherty. *An Introduction to Morphological Image Processing*. SPIE Optical Engineering Press, 1992. Google-Books-ID: 1kvxAAAAMAAJ.

[DSM12] Macam S. Dattathreya, Harpreet Singh, and Thomas Meitzler. Detection and Elimination of a Potential Fire in Engine and Battery Compartments of Hybrid Electric Vehicles. *Advances in Fuzzy Systems*, 2012:e687652, 2012.

[DW84] K. W. Dickinson and R. C. Waterfall. IMAGE PROCESSING APPLIED TO TRAFFIC, 1-A GENERAL REVIEW. *Traffic Engineering & Control*, 25(1), January 1984.

[FPWW03] R. Fisher, S. Perkins, A. Walker, and Wolfart. Morphology, 2003.

[FRBR09] M. C. Figueiredo, R. J. F. Rossetti, R. A. M. Braga, and L. P. Reis. An approach to simulate autonomous vehicles in urban traffic scenarios. In *2009 12th International IEEE Conference on Intelligent Transportation Systems*, pages 1–6, October 2009.

[GBBV16] C. Grana, F. Bolelli, L. Baraldi, and R. Vezzani. YACCLAB - Yet Another Connected Components Labeling Benchmark. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 3109–3114, December 2016.

[GBC10] C. Grana, D. Borghesani, and R. Cucchiara. Optimized Block-Based Connected Components Labeling With Decision Trees. *IEEE Transactions on Image Processing*, 19(6):1596–1609, June 2010.

[GMMP02] S. Gupte, O. Masoud, R. F. K. Martin, and N. P. Papanikolopoulos. Detection and classification of vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 3(1):37–47, March 2002.

[Got10] Siegfried Gottwald. An early approach toward graded identity and graded membership in set theory. *Fuzzy Sets and Systems*, 161(18):2369–2379, September 2010.

[GW92] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. 1992.

[HBJ+16] K. Hammoudi, H. Benhabiles, A. Jandial, F. Dornaika, and J. Mouzna. Self-driven and direct spatio-temporal mechanisms for the vision-based parking slot surveillance. In *2016 SAI Computing Conference (SAI)*, pages 1327–1329, July 2016.

[HK12] M. F. Hashmi and A. G. Keskar. Analysis and monitoring of a high density traffic flow at T-intersection using statistical computer vision based approach. In *2012 12th International Conference on Intelligent Systems Design and Applications (ISDA)*, pages 52–57, November 2012.

[Hua96] T. Huang. Computer Vision : Evolution And Promise, 1996.

[JAG08] G. Jun, J. K. Aggarwal, and M. Gokmen. Tracking and Segmentation of Highway Vehicles in Cluttered and Crowded Scenes. In *2008 IEEE Workshop on Applications of Computer Vision*, pages 1–6, January 2008.

[JBS14] J. P. Jodoin, G. A. Bilodeau, and N. Saunier. Urban Tracker: Multiple object tracking in urban mixed traffic. In *IEEE Winter Conference on Applications of Computer Vision*, pages 885–892, March 2014.

[Jen15] Niall Jenkins. 245 million video surveillance cameras installed globally in 2014 - IHS Technology, 2015.

[Kla67] Dieter Klaua. Ein Ansatz zur mehrwertigen Mengenlehre. *Mathematische Nachrichten*, 33(5-6):273–296, January 1967.

[Koz17] Michael Koziol. 'World first': Government moves to radically overhaul Australia's international airports. *The Sydney Morning Herald*, January 2017.

[Lab17] Virtual Labs. Theory - Virtual Lab in Image Processing, 2017.

[LB78] Michael L. Baird. SIGHT-I: A Computer Vision System for Automated IC Chip Manufacture. *IEEE Transactions on Systems, Man, and Cybernetics*, 8(2):133–139, February 1978.

[LBT17] Robert P. Loce, Raja Bala, and Mohan Trivedi. Detection of Passenger Compartment Violations. In *Computer Vision and Imaging in Intelligent Transportation Systems*, pages 432–. Wiley-IEEE Press, 2017.

[LCC12] X. Li, Q. Chen, and H. Chen. Detection and tracking of moving object based on PTZ camera. In *2012 IEEE Fifth International Conference on Advanced Computational Intelligence (ICACI)*, pages 493–497, October 2012.

[LKR+16] G. Lira, Z. Kokkinogenis, R. J. F. Rossetti, D. C. Moura, and T. Rúbio. A computer-vision approach to traffic analysis over intersections. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 47–53, November 2016.

[LRB09] P. F. Q. Loureiro, R. J. F. Rossetti, and R. A. M. Braga. Video processing techniques for traffic information acquisition using uncontrolled video streams. In *2009 12th International IEEE Conference on Intelligent Transportation Systems*, pages 1–7, October 2009.

[LV01] B. P. L. Lo and S. A. Velastin. Automatic congestion detection system for underground platforms. In *Proceedings of 2001 International Symposium on Intelligent Multimedia, Video and Speech Processing. ISIMP 2001 (IEEE Cat. No.01EX489)*, pages 158–161, 2001.

[MCKT00] I. Mikic, P. C. Cosman, G. T. Kogut, and M. M. Trivedi. Moving shadow and object detection in traffic scenes. In *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, volume 1, pages 321–324 vol.1, 2000.

[MH80] D. Marr and E. Hildreth. Theory of edge detection. *Proceedings of the Royal Society of London. Series B, Biological sciences*, 207 1167:187–217, 1980.

[MH07] J. Martín-Herrero. Hybrid object labelling in digital images. *Machine Vision and Applications*, 18(1):1–15, 2007.

[Nav00] Navigant. Transportation Forecast: Light Duty Vehicles, 2017-04-18T20:30:25+00:00.

[Ope17a] OpenCV. About - OpenCV library, 2017.

[Ope17b] OpenCV. OpenCV: Cv::SimpleBlobDetector Class Reference, 2017.

[PB12] Debasish Pal and Debasish Bhattacharya. Effect of Road Traffic Noise Pollution on Human Work Efficiency in Government Offices, Private Organizations, and Commercial Business Centres in Agartala City Using Fuzzy Expert System: A Case Study. *Advances in Fuzzy Systems*, 2012:e828593, 2012.

[Pic04] M. Piccardi. Background subtraction techniques: A review. In *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583)*, volume 4, pages 3099–3104 vol.4, October 2004.

[PMGT01] A. Prati, I. Mikic, C. Grana, and M. M. Trivedi. Shadow detection algorithms for traffic flow analysis: A comparative study. In *ITSC 2001. 2001 IEEE Intelligent Transportation Systems. Proceedings (Cat. No.01TH8585)*, pages 340–345, 2001.

[PRK11] L. S. Passos, R. J. F. Rossetti, and Z. Kokkinogenis. Towards the next-generation traffic simulation tools: A first appraisal. In *6th Iberian Conference on Information Systems and Technologies (CISTI 2011)*, pages 1–6, June 2011.

[PRO10] L. S. Passos, R. J. F. Rossetti, and E. C. Oliveira. Ambient-centred intelligent traffic control and management. In *13th International IEEE Conference on Intelligent Transportation Systems*, pages 224–229, September 2010.

[QGJX15] J. Qianyin, L. Guoming, Y. Jinwei, and L. Xiying. A model based method of pedestrian abnormal behavior detection in traffic scene. In *2015 IEEE First International Smart Cities Conference (ISC2)*, pages 1–6, October 2015.

[RBH90] A. Rourke, M. G. H. Bell, and N. Hoose. Road traffic monitoring using image processing. In *Third International Conference on Road Traffic Control, 1990.*, pages 163–167, May 1990.

[Rep17] ReportLinker. Global Video Analytics Market 2017-2021 - market research report, 2017.

[SJK09] Y. Sheikh, O. Javed, and T. Kanade. Background Subtraction for Freely Moving Cameras. In *2009 IEEE 12th International Conference on Computer Vision*, pages 1219–1225, September 2009.

[SLZ16] C. Supriyanto, A. Luthfiarta, and J. Zeniarja. An unsupervised approach for traffic sign recognition based on bag-of-visual-words. In *2016 8th International Conference on Information Technology and Electrical Engineering (ICITEE)*, pages 1–4, October 2016.

[SMG+93] J. W. Snell, M. B. Merickel, J. C. Goble, J. B. Brookeman, and N. F. Kassell. Model-based segmentation of the brain from 3-D MRI using active surfaces. In *1993 IEEE Annual Northeast Bioengineering Conference*, pages 164–165, March 1993.

[Sob89] Irwin Sobel. An Isotropic 3 3 Image Gradient Operator (PDF Download Available), 1989.

[SRM+16] M. Sandim, R. J. F. Rossetti, D. C. Moura, Z. Kokkinogenis, and T. R. P. M. Rúbio. Using GPS-based AVL data to calculate and predict traffic network performance metrics: A systematic review. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1692–1699, November 2016.

[SS01] Linda G. Shapiro and George C. Stockman. *Computer Vision*. Prentice Hall, Upper Saddle River, NJ, 2001.

[SS11] D. Soendoro and I. Supriana. Traffic sign recognition with Color-based Method, shape-arc estimation and SVM. In *Proceedings of the 2011 International Conference on Electrical Engineering and Informatics*, pages 1–6, July 2011.

[ST94] Jianbo Shi and C. Tomasi. Good features to track. In *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, June 1994.

[Sto80] K. J. Stout. Computer vision and robots. *Production Engineer*, 59(4):9–, April 1980.

[TARO10] I. J. P. M. Timóteo, M. R. Araújo, R. J. F. Rossetti, and E. C. Oliveira. TraSMAPI: An API oriented towards Multi-Agent Systems real-time interaction with multiple Traffic Simulators. In *13th International IEEE Conference on Intelligent Transportation Systems*, pages 1183–1188, September 2010.

[Wik17] Wikipedia. Connected component labeling, June 2017. Page Version ID: 783324000.

[WLGC16] S. Wang, F. Liu, Z. Gan, and Z. Cui. Vehicle type classification via adaptive feature clustering for traffic surveillance video. In *2016 8th International Conference on Wireless Communications Signal Processing (WCSP)*, pages 1–5, October 2016.

[Zad65] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, June 1965.

[Ziv04] Z. Zivkovic. Improved adaptive Gaussian mixture model for background subtraction. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 2, pages 28–31 Vol.2, August 2004.

[ZRC14] A. Zaiat, R. J. F. Rossetti, and R. J. S. Coelho. Towards an integrated multimodal transportation dashboard. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 145–150, October 2014.

[ZY14] Daniya Zamalieva and Alper Yilmaz. Background subtraction for the moving camera: A geometric approach. *Computer Vision and Image Understanding*, 127:73–85, October 2014.